CASEVision™/ClearCase Reference Pages

Document Number 007-1613-030

CONTRIBUTORS

Written by John Posner
Illustrated by John Posner
Production by Gloria Ackley
Engineering contributions by Atria Software, Inc.
Cover design and illustration by Rob Aguilar, Rikk Carey, Dean Hodgkinson, Erik Lindholm, and Kay Maitz

CASEVision™/ClearCase Reference Pages
Document Number 007-1613-030

Contents

cleartool Manual Pages

Non-cleartool Manual Pages

Permuted Index

Figures

Tables

Introduction

ClearCase™ is a comprehensive *software configuration management* system from Atria™ Software, Inc. It manages multiple variants of evolving software systems, tracks which versions were used in software builds, performs builds of individual programs or entire releases according to user-defined version specifications, and enforces site-specific development policies.

This manual describes in detail the ClearCase facilities with which you perform version control and configuration management tasks. It focuses on command syntax and use, and is not intended to be a learning tool. It is a reference manual, which will be most useful if you have already obtained background about ClearCase through other means.

If you are not already familiar with basic ClearCase concepts and use, you should first refer to the learning documents supplied with your system, including the *ClearCase Concepts Manual*, *Getting Started with ClearCase*, and the *ClearCase User's Manual*. If your responsibilities include software installation, licensing, and other administrative tasks, you should also refer to the *ClearCase Administrator's Manual* and the *ClearCase Notebook*.

Organization of this Manual

This manual does *not* use the standard "UNIX Reference Manual" organization: Section 1, Section 1M, Section 4, and so on. Instead, it is organized as follows:

Part 1: **Manual pages for *cleartool* and its subcommands**
The *cleartool* program is the principal command-line interface (CLI) to ClearCase capabilities. It has many subcommands, such as *mkelem* ("make element") and *mklbtype* ("make label type"). This part of the manual includes an overall *cleartool* manual page, followed by separate manual pages for each subcommand, organized alphabetically. The page headings of these manual pages use a sans-serif font for the name:

Part 2: **All other ClearCase manual pages**
All other manual pages are included in this section, in alphabetical order. Manual pages for programs, data structures, and miscellany are intermixed. The page headings of these manual pages use a serif font for the name:

Part 3: **Permuted index**
A standard-UNIX "permuted index", constructed from all the ClearCase manual pages.

Typographical Conventions

This manual uses certain typographical conventions. Within the SYNOPSIS section that begins most manual pages:

- **Boldface** type is used for command names and command options.
- If a command or option name has a short form, a "medial dot" (·) character indicates the shortest legal abbreviation. For example:

  lsc·heckout

  This means that you can truncate the command name to **lsc** or any of its intermediate spellings (**lsch**, **lsche**, **lschec**, and so on).
- Square brackets (**[** ... **]**) around an option indicate that it is not required.
- *Italic* type is used for command arguments. For example:

  mktag [–rep/ lace] *view-storage-dir-pname view-tag*
- Braces (**{** ... **}**) around a group of options mean that exactly one of the options is required.
- An ellipsis (**...**) means that the preceding option or argument can be repeated as many times as desired.

NOTE: In certain contexts, ClearCase recognizes "**...**" within a pathname as a wildcard, similar to "*" or "?".

Throughout the manual pages, the following conventions also apply:

- ClearCase commands, UNIX commands, and *cleartool* subcommands appear in *italic* type:
  - For example, using *mkelem* to create a new element.
- *G*lossary terms and document names also appear in *italic* type:
  - the versions of elements selected by the view's *config spec*
  - see the *ClearCase Administrator's Manual* for a step-by-step procedure
- Simple file names, pathnames, and other user-supplied names also appear in *italic* type:
  - On most hosts, the registry directory contains only the *rgy_hosts.conf* and *rgy_region.conf* configuration files
  - the version to which version label *REL2* has been attached

- Examples, system output, prompts, and messages appear in `monospace` type. Within an extended example, user input appears in a contrasting font:
  - `% cleartool mklabel BL6 util.c`
    `Created label "BL6" on "util.c" version "main/1".`
- Names of keyboard characters appear in `monospace` type.
  - press `<Return>` after entering the file name

NAME cleartool_divider – separator page

DESCRIPTION

FOR POSITION ONLY

THIS PAGE TO BE REPLACED

BY A FULL-PAGE RUBYLITH AND

"cleartool Manual Pages"

FOR POSITION ONLY

BLANK PAGE WITH A

FULL-PAGE RUBYLITH

NAME cleartool – ClearCase user-level commands (command-line interface)

SYNOPSIS

- Single-command mode:

  cleartool *subcommand* [*options/args*]

- Interactive mode:

  % **cleartool**
  cleartool> *subcommand* [*options/args*]
  ⋮
  cleartool> **quit**

DESCRIPTION

cleartool is the primary command-line interface to ClearCase's version-control and configuration management software. It has a rich set of subcommands that create, modify, and manage the information in ClearCase VOBs and views.

CLEARTOOL SUBCOMMANDS

Manual pages for the individual *cleartool* subcommands make up the first section of this manual:

annotate	ls	mkelem	recoverview	rmview
apropos	lscheckout	mkeltype	reformatview	rmvob
catcr	lsdo	mkhlink	reformatvob	rnpool
catcs	lshistory	mkhltype	register	rntype
cd	lslock	mklabel	reserve	setcs
checkin	lspool	mklbtype	rmattr	setview
checkout	lsprivate	mkpool	rmbranch	shell
chevent	lsreplica	mktag	rmdo	space
chpool	lstype	mktrigger	rmelem	startview
chtype	lsview	mktrtype	rmhlink	umount
describe	lsvob	mkview	rmlabel	uncheckout
diff	lsvtree	mkvob	rmmerge	unlock
diffcr	man	mount	rmname	unregister
edcs	merge	mv	rmpool	unreserve
find	mkattr	protect	rmtag	winkin
findmerge	mkattype	protectvob	rmtrigger	xdiff
help	mkbranch	pwd	rmtype	xlsvtree
ln	mkbrtype	pwv	rmver	xmerge
lock	mkdir	quit		

NOTE: The *lstag* command is supported for compatibility with older ClearCase releases; it is essentially an obsolete version of the *lsview* command.

GETTING HELP

cleartool provides several on-line help facilities for its subcommands:

- **Syntax summary** — To display a syntax summary for an individual subcommand, use the *help* subcommand or the `–help` option:

```
cleartool help                          (syntax of all subcommands)
cleartool help mklabel                  (syntax of one subcommand)
cleartool mklabel -help                 (syntax of one subcommand)
```

- **Manual pages** — *cleartool* has its own interface to the UNIX *man*(1) command. Enter `cleartool man` *command_name* to display the manual page for a subcommand.
- **Permuted index** — File */usr/atria/doc/man/permuted_index* contains the same information as the permuted index printed in this manual.
- **'whatis' file** — Use the *apropos* subcommand to extract help information from file */usr/atria/doc/man/whatis*, which is in the standard UNIX "whatis" format. This subcommand can also extract auxiliary information — for example, glossary entries — from file */usr/atria/doc/man/whatis.aux*.

For a master Table of Contents to all ClearCase manual pages, including those for the *cleartool* subcommands, see the *clearcase* manual page: `cleartool man clearcase`. In addition, see the "Permuted Index" in this manual.

USAGE OVERVIEW

You can use *cleartool* in either *single-command mode* or *interactive mode*. A single *cleartool* command can be invoked from the shell using this syntax:

cleartool *subcommand* [*options-and-args*]

If you wish to enter a series of subcommands, enter the *cleartool* command with no arguments. This places you at the *interactive mode* prompt:

```
cleartool>
```

You can then issue any number of subcommands (simply called "commands" from now on), ending with *quit* to return to the shell. *cleartool* commands can be continued onto additional lines with the backslash (\) character, as with UNIX shells.

Command options may appear in any order, but all options must precede any non-option arguments (typically, names of files, versions, branches, and so on). If an option is followed by an additional *argument*, such as `-branch /main/bugfix`, there must be white space between the option string and the argument. If the argument itself includes space characters, it must be quoted.

Command Abbreviations and Aliases

Many subcommand names and option words can be abbreviated. A subcommand's syntax summary indicates all valid abbreviations. For example:

–pre·decessor

This means that you can abbreviate the option to the minimal "–pre", or to any intermediate spelling: "–pred", "–prede", and so on.

For option words, the minimal abbreviation is always three characters or fewer.

A few *cleartool* commands have a built-in command alias. For example, *checkin*'s alias is *ci*; *checkout*'s alias is *co*. These commands are equivalent:

```
% cleartool checkin test.c
% cleartool ci test.c
```

PATHNAMES IN CLEARTOOL COMMANDS

Many *cleartool* commands require a pathname as an argument, such as the name of a file element, directory element, or view-private file. You can use either kind of standard UNIX pathname: *full* or *relative*. In many cases, you can also use a ClearCase-defined variant: a *view-extended* pathname (full or relative) or a *version-extended* pathname (full or relative).

A *full pathname* begins with a slash (`/`). For example:

`/usr/src/project`	*full pathname*
`/usr/bin/cc`	*full pathname*
`/view/jpb/usr/src/project/test.c`	*view–extended full pathname*
`/usr/src/project@@/main/3/test.c/main/bugfix/4`	*version–extended full pathname*

A *relative pathname* does not begin with a slash. For example:

`test.c`	*relative pathname*
`../lib`	*relative pathname*
`motif/libX.a`	*relative pathname*
`../../beta_vu/usr/src/project`	*view–extended relative pathname*
`test.c@@/main/4`	*version–extended relative pathname*

For both full or relative pathnames:

- The standard UNIX pathname of an element implicitly references the version selected by the current view. (This feature is called *transparency*.)
- A view-extended pathname references the version of the element selected by the specified view.
- A version-extended pathname directly references a particular version in an element's version tree.

For more information, see the *version_selector* and *pathnames_ccase* manual pages.

PROCESSING OF VOB SYMBOLIC LINKS

In general, *cleartool* commands do not traverse VOB symbolic links; rather, they operate on the link objects themselves. For example:

- You cannot perform a *checkout* command on a VOB symbolic link, even if it points to an element.
- A *describe* command lists information on a VOB symbolic link object, not on the object to which it points.
- A `mklabel -recurse` command walks the entire subtree of a directory element, but it does not traverse any VOB symbolic links it encounters.

COMMAND-LINE PROCESSING

In single-command mode, the *cleartool* command you enter is first processed by the UNIX shell. The shell expands file name patterns and environment variables, and it interprets quotes and other special characters. *cleartool* processes the resulting argument list directly, without any further interpretation.

In interactive mode, *cleartool* itself interprets the command line similarly, but not identically, to the UNIX shells:

line continuation
: A `\<NL>` sequence is replaced by a `<Space>` character.

character escape
: The two-character sequence `\` *special-char* suppresses the special meaning of the character.

single-quoting
: Allows white space characters and other special characters to be included in command argument. Within a single-quoted string (`'` ... `'`), a double-quote character has no special meaning, and `\'` is replaced by `'`.

double-quoting
: Allows white space characters and other special characters to be included in command argument. Within a double-quoted string (`"` ... `"`), `\"` is replaced by `"`, and `\'` is replaced by `'`.

commenting
: Command lines that begin with a pound sign (#) character are ignored.

wildcards
: File name patterns (including *, ?, and so on) that are not enclosed in quotes are expanded as described in the *wildcards_ccase* manual page. These patterns are also supported in config specs and, except for ellipsis (`...`), by the UNIX shells. (The meaning of ellipsis is slightly different in config specs; see the *config_spec* manual page.)

In interactive mode, *cleartool* does *not* expand environment variables and does *not* perform command substitution (`‘` ... `‘`).

EVENT RECORDS AND COMMENTS

Each change to a VOB (checkin of new version, attaching of a version label, and so on) is accompanied by the creation of an *event record* in the VOB database. Many *cleartool* commands allow you to annotate the event record(s) they create with a comment string. Commands that display event record information (*describe*, *lscheckout*, *lshistory*, *lslock*, *lspool*, *lsreplica*, and *lstype*) show the comments, as well. See the *fmt_ccase* manual page for a description of the report-writing facility built into these commands.

All commands that accept comment strings recognize the same options:

−c *comment-string*
: Specifies a comment for all the event records. The comment string must be a single command-line token; typically, you must quote it.

–cq Prompts for one comment, to be placed in all the event records created by this command.

–cqe For each object processed by this command, prompts for a comment to be placed in the corresponding event record.

–nc ("no additional comment") For each object processed by this command, creates an event record with no user-supplied comment string.

A `–cq` or `–cqe` comment string can span several lines; end it by typing an `EOF` character (typically, `<Ctrl-D>`), or by entering a line that consists of a single period (`.`) character.

The *chevent* command revises the comment string in an existing event record. See the *events_ccase* manual page for a detailed discussion of event records.

Customizing Comment Handling

Each command that accepts a comment string has *comment default*, which takes effect if you enter the command without any comment option. For example, the *checkin* command's comment default is `–cqe`, causing *cleartool* to prompt you to enter a comment for each element being checked in. The *ln* command's comment default is `–nc`: create the event record without a comment.

You can customize *cleartool*'s comment-handling with a *user profile* file, *.clearcase_profile* in your home directory. For example, you might establish `–cqe` as the comment default for the *ln* command. See the *profile_ccase* manual page for details.

PERMISSIONS CHECKING

All *cleartool* commands that modify ("write") a VOB are subjected to permissions checking. The following hierarchy is used, in a command-specific manner, to determine whether a command should proceed or be cancelled:

- the *root* user (superuser)
- the VOB owner (that is, the user who created the VOB storage area)
- the owner of the corresponding element (for modifications to branches and versions)
- the creator of the type object (for modifications to objects of that type)
- the creator of a particular version or derived object
- members of an element's group or derived object's group (same UNIX group ID)

For example, the *root* user always has permission to use commands that modify a VOB. However, if you try to modify an element that you do not own, and are neither the VOB owner nor the *root* user, *cleartool* will not allow the operation to proceed.

Each *cleartool* command description lists the permissions required for using the command. The *chtype* command, for example, lists these requirements for changing an element type:

```
element owner, vob owner, root user
```

This means that you must be the owner of the element whose type is to be changed, the owner of that element's VOB, or *root*. Otherwise, *cleartool* will not allow the *chtype* operation to proceed.

ClearCase also provides for temporary access control through explicit *locking* of individual objects with the *lock* command. When an object is locked, it cannot be modified by anyone (except those explicitly excluded from the lock), even *root*, the VOB owner, and the user who created the lock.

cleartool command descriptions list the locks that can prevent a command from being executed, even if you have the necessary permissions. For example, the *chtype* command lists three locks that would prevent you from changing an element type:

```
VOB, element type, pool (non-directory elements only)
```

This means that *chtype* would fail if the VOB containing the element were locked, if the element's type were locked (such as the *text_file* type), or the storage pool containing the (non-directory) element were locked.

EXIT STATUS

If you exit *cleartool* by entering a *quit* command in interactive mode, the exit status is 0. The exit status from single-command mode depends on whether the command succeeded (zero exit status) or generated an error message (nonzero exit status).

Note that for the *diff* command, "success" means finding no differences.

FILES

/usr/atria/doc/man/permuted_index	permuted index
/usr/atria/doc/man/whatis	whatis file
/usr/atria/doc/man/whatis.aux	auxiliary whatis file

SEE ALSO

clearcase, config_spec, ct_permissions, events_ccase, fmt_ccase, profile_ccase, version_selector

NAME ct_permissions – access permissions for cleartool commands

DESCRIPTION

This manual page summarizes the access permissions needed to use *cleartool* subcommands. In general, only subcommands that modify a VOB are subjected to permissions checking. *cleartool* uses the following permissions hierarchy (most-privileged to least-privileged):

- *root* user (superuser)
- VOB owner (initially set to the user who created the VOB storage area)
- owner of the relevant element (for modifications to branches and versions)
- creator of the relevant type (for modifications to type objects)
- creator of a particular version or derived object
- creator of a particular storage pool
- user associated with a particular event
- members of an element's or derived object's group (same UNIX-level GID)

The sections below list all *cleartool* subcommands, categorized by their permissions requirements.

Permissions: none

annotate	lslock	mkview <3>
apropos	lspool	mkvob <3>
catcr	lsprivate	mv <2>
catcs	lsreplica	pwd
cd	lstype	pwv
describe	lsview	quit
diff	lsvob	rmname <2>
diffcr	lsvtree	rmtag
edcs	man	rmview
find	mkattype	setcs
findmerge <1>	mkbrtype	setview
help	mkdir <2>	shell
ln <2>	mkelem <2>	startview
ls	mkeltype	winkin
lscheckout	mkhltype	xdiff
lsdo	mklbtype	xlsvtree
lshistory	mktag	

NOTE 1: no permissions required for "search" functionality
NOTE 2: one or more directory elements must be checked out
NOTE 3: standard UNIX permissions for creating a subdirectory required

Permissions: element group member, element owner, VOB owner, root user

checkout
merge <1>
mkattr
mkbranch
mkhlink
mklabel
mktrigger
reserve
rmattr
rmhlink
rmlabel
rmmerge
rmtrigger
unreserve
xmerge <1>

NOTE 1: applies to creation of merge arrows only, not to data

Permissions: version creator, element owner, VOB owner, root user

checkin
rmver
uncheckout

Permissions: VOB owner, root user

chpool
lock (pool or VOB)
mkpool
mktrtype
reformatvob
rmvob
unlock (pool or VOB)

Permissions: element owner, VOB owner, root user

chtype (element type)
lock (element)
protect (element or derived object)
rmelem
unlock (element)

Permissions: user associated with event, VOB owner, root user

chevent

Permissions: branch creator, element owner, VOB owner, root user

chtype (branch type)
lock (branch)
rmbranch
unlock (branch)

Permissions: type creator, VOB owner, root user

lock (type object)
mkattype -replace
mkbrtype -replace

mkeltype -replace
mkhltype -replace
mklbtype -replace
mktrtype -replace
rmtype
rntype
unlock (type object)

Permissions: user associated with event, VOB owner, root user

chevent

Permissions: DO owner, VOB owner, root user

protect (derived object)

Permissions: pool creator, VOB owner, root user

rmpool
rnpool

Permissions: DO group member, DO owner, VOB owner, root user

rmdo

NOTE: Only the VOB owner and the *root* user can delete a shared derived object.

Permissions: root user

protectvob

SEE ALSO

individual *cleartool* subcommand descriptions
clearcase, cleartool

NAME annotate – annotate lines of text file / timestamps, usernames, etc.

SYNOPSIS

ann·otate [**–all** | **–rm**] [**–nco**] [**–out** *pname*] [**–for·ce**]
[**–nda·ta**] [**–nhe·ader**]
[**–s·hort** | **–l·ong** | **–fmt** *format*[**,***hdr-format*[**,***elide-format*]]]
[**–rmf·mt** *rm-format*]
pname …

DESCRIPTION

Lists the contents of a version, annotating each line to indicate when, and in which version, the line was added. You can customize the annotations using ClearCase's report-writing facility (`–fmt` option), which is described in the *fmt_ccase* manual page. By default, *annotate* writes its output to a file with the `.ann` extension. You can send output to *stdout*, or to an arbitrary file, with the `–out` option.

RESTRICTIONS: See "Type Manager Interface" below.

Line of Descent

Each version has a *line of descent* (Figure 1), a sequence of ancestor versions going all the way back to */main/0*. *annotate*'s default listing has a header section that includes the event records of all the versions in the line of descent of the annotated version.

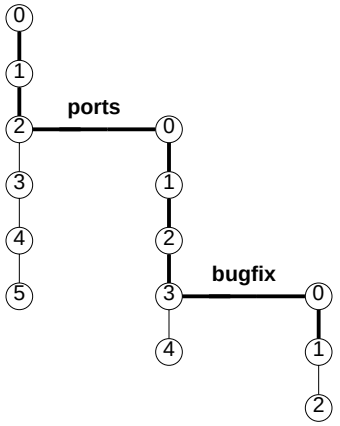


Figure 1. Line of Descent of a Version

Type Manager Interface

The *annotate* command extracts information from the element's versions. To do so, it invokes the *annotate* method of the element's type manager. Only the *text_file_delta* and *z_text_file_delta* type managers (which correspond to the predefined element types *text_file* and *compressed_text_file*) include an *annotate* method. You must use the `–ndata` option when annotating versions of other element types.

REPORT FORMAT

The default *annotate* report format includes the following components, as shown in Figure 2.

- A *header section*, which lists the event record for each version along the line of descent, in standard *lshistory* format.
- *Text line annotations*, which include a *bar graph* indicating how long ago the line first appeared in an ancestor version, along with that version's timestamp, creator, and version-ID.
- *Elision strings*, which replace text line annotations that would duplicate the annotation on the preceding line. An elision string includes the bar graph and a single dot (`.`) character.
- *Source lines* from the specified version. Any `<Tab>` characters in source lines are expanded according to the value of environment variable CLEARCASE_TAB_SIZE (default: 8).

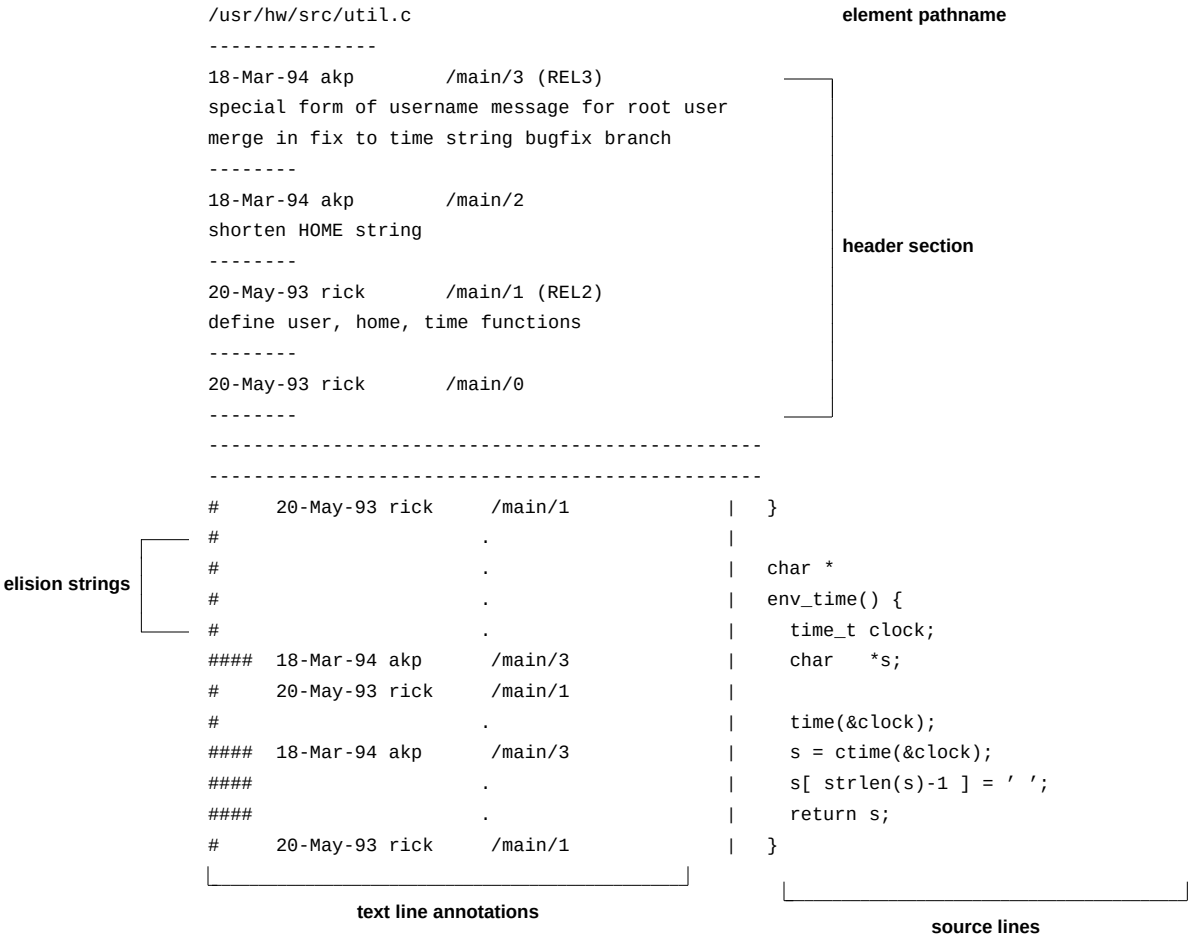


Figure 2. Default ´annotate´ Report Format

If you use the –rm or –all option, the report also includes *deletion annotations*. These appear on text lines that are not in the annotated version, but do exist in some other version of the element:

```
#     20-May-93 rick     /main/1    |                    |
#                        .          |                    |     time(&clock);
#                        .          | DEL 18-Mar-94 akp  |     return ctime(&clock);
```

OPTIONS AND ARGUMENTS

Including Other Text Lines. *Default: annotate*'s listing includes only text lines that are present in the specified version.

–rm Also includes "removed lines" — text lines that were present in one or more versions along the line of descent, but do not occur in the specified version. See also the –rmfmt option.

–all Expands the listing to include all text lines that occurred in *any* version of the element, including lines in versions that are not along the line of descent. (Lines from versions outside the line of descent are annotated as UNRELATED; this annotation appears in the same column used to annotate deletion lines.)

Handling of Checked-Out Versions. *Default:* An error occurs if you specify a checked-out version. (The type manager can annotate checked-in versions only.)

–nco If you specify a checked-out version, *annotate* automatically uses the version from which it was checked out.

Destination of Listing. *Default:* Command output is sent to the file *input-file*.ann.

–out *output-pname*

If *output-pname* is a file name, redirects command output to the specified file (overwriting the file if it already exists). If *output-pname* is a single hyphen character (-out -), sends command output to *stdout*. If *output-pname* is a directory, places command output for each annotated version in a file within that the directory (which must already exist).

If you use this option when annotating more than one version, *output-pname* must be a directory.

Report Format. *Default:* The source file is listed as described in section "Report Format" above.

–s·hort Uses predefined annotation format strings that yield an abbreviated report.

–l·ong Uses predefined annotation format strings that yield a verbose report.

–fmt *format*[**,***hdr-format*[**,***elide-format*]]

Specifies a display *format* for "primary" annotations, and optionally, for the header section and/or elision strings. Format strings must be quoted. The default *format* is "%BAd %Sd %-8.8u %-16.16Vn | ".

Use a hyphen (-) to designate a default format string. For example, to supply a *hdr-format*, but not a primary annotation *format*, use the construction -fmt -,*hdr-format*. It is usually desirable to terminate the *hdr-format* with a <NL> character, by using \n.

If you omit the *elide-format*, it is computed based on the primary line-by-line annotation: all characters except `<Tab>` and `|` in the primary annotation are replaced by `<Space>`, and the middle character, if it is a `<Space>`, is replaced by a period (.).

In general, it is simpler to use fixed-width fields, not tab-character specifiers (`\t`), to create aligned columns of annotations. See the *fmt_ccase* manual page for more details on composing format strings.

–rmf·mt *rm-format*
Specifies a format for deletion annotations (see also `–rm` and `–all`). The default format is `"DEL %Sd %-8.8u | "`.

–for·ce Displays each text-line's annotation, even if it duplicated the previous line's annotation. This option suppresses use of elision strings.

Partial Reports. *Default:* The report includes both a header section and the annotated text lines.

–nda·ta Suppresses the annotated text lines; the report consists of the header section only.

–nhe·ader Suppresses the header section; the report consists of the annotated text lines only.

EXAMPLES

- Annotate a source file, using the short format.

```
% cleartool annotate –short msg.c
Annotated result written to "msg.c.ann".

% cat msg.c.ann
/usr/vobs/src/msg.c
---------------
24-Apr-94 anne       /main/rel2_bugfix/9
12-Mar-94 ravi       /main/rel2_bugfix/8
  .
  .
  .
23-Apr-94 rks        /main/48 (REL2)
20-Apr-94 spc        /main/47
  .
  .
  .
-----------------------------------------------
-----------------------------------------------
20-May-93 | #include "hello.h"
      .   |
      .   |   char *
21-Apr-94 |   env_user() {
      .   |     char * user_env;
      .   |     user_env = getenv("USER");
  .
  .
  .
      .   |     time_t clock;
24-Mar-94 |     char   *s;
20-Sep-93 |
14-Jun-94 |     s = ctime(&clock);
      .   |     s[ strlen(s)-1 ] = ' ';
      .   |     return s;
20-May-93 |   }
```

- Annotate a source file, using the long format.

```
% cleartool annotate –long msg.c
Annotated result written to "msg.c.ann".
```

```
% cat msg.c.ann
/vobs/src/msg.c
---------------
02-Apr-94.10:51:54 ##### Steve (scd.user@reach)  /main/rel2_bugfix/1
a test
  .
  .
------------------------------------------------
------------------------------------------------
##### 01-Apr-94.16:19:25 scd      /main/1               | #include "hello.h"
##### 02-Apr-94.10:51:54 scd      /main/rel2_bugfix/1   | /* a test*/
##### 01-Apr-94.16:19:25 scd      /main/1               |
#####           .                                      |   char *
#####           .                                      |   hello_msg() {
  .
  .
```

- Annotate a source file, and write the output to *stdout*. Display deletion lines, customize the annotation format, and suppress the header output.

```
% cleartool annotate –out – –fmt "%Sd %–8.8u | " –rm –nheader util.c
20-May-93 anne     |                      | #include "hello.h"
        .          |                      |
        .          |                      |   char *
        .          |                      |   env_user() {
        .          | DEL 08-Feb-94 gcd    |     return getenv("USER");
08-Feb-94 gcd      |                      |     char *str = getenv("USER");
        .          |                      |     if ( strcmp(str,"root") == 0 )
  .
  .
```

- Customize the header format, but use the default format for text line annotations.

```
% cleartool annotate –out – –fmt –,"Version %Vn created by %u.\n" util.c
version /main/3 created by anne.
version /main/2 created by anne.
version /main/1 created by rick.
version /main/0 created by rick.
------------------------------------------------
------------------------------------------------
#     20-May-93 rick     /main/1            | #include "hello.h"
#                        .                  |
#                        .                  |   char *
#                        .                  |   env_user() {
###   08-Feb-94 anne     /main/3            |     char *str = getenv("USER");
###                      .                  |     if ( strcmp(str,"root") == 0 )
  .
  .
```

SEE ALSO

type_manager, fmt_ccase

NAME apropos – display cleartool command summary information

SYNOPSIS

- Extract information from standard 'whatis' file:

  apr·opos *topic …*

- Extract information from auxiliary 'whatis' file:

  apr·opos [**–glo·ssary**] *topic-args*

DESCRIPTION

Extracts information from file *doc/man/whatis* in your host's ClearCase installation area (by default, */usr/atria*). Use *apropos* as you would use the standard UNIX *whatis*(1) or *apropos*(1) command. Alternatively, use the `–glossary` option to extract a glossary definition or other help information from the auxiliary "whatis" file.

OPTIONS AND ARGUMENTS

Default: A lookup is performed in the standard ClearCase whatis file.

topic … *apropos* makes a separate search for each *topic* character string in the standard ClearCase whatis file. The string can occur anywhere within the line.

–glo·ssary [*topic-args*]
All of the arguments are combined into a single character string; *apropos* displays all sections in the auxiliary ClearCase whatis file whose header lines include this character string. Omitting the *topic-args* causes the entire auxiliary file to be displayed.

EXAMPLES

- Search for lines with the word "reserve" in the standard ClearCase whatis file.

```
% cleartool apropos reserve
reserve             - convert an unreserved checkout to reserved
unreserve           - change a reserved checkout to unreserved
```

- Search in the auxiliary ClearCase whatis file for glossary terms that include the string "reserve".

```
% cleartool apropos –glossary reserve
+ reserve state
        For a checked-out version, either "reserved" or "unreserved". The
        reserve and unreserve commands change the reserve state of a
        checked-out file.

+ reserved checkout
        See checkout.

+ unreserved checkout
        See checkout.
```

- Search in the auxiliary ClearCase whatis file for glossary terms that include the phrase “derived object”.

```
% cleartool apropos –glossary derived object
+ configuration (of a derived object)
        The information recorded in a derived object’s CR: versions of source
        files used to build the object, build script, build options, and so
        on.

+ degenerate derived object
        A derived object that cannot be successfully processed, because it
        data container and/or associated configuration record are not
        available.

  ⋮
```

FILES

/usr/atria/doc/man/whatis
/usr/atria/doc/man/whatis.aux

SEE ALSO

cleartool subcommands: help, man

NAME catcr – display configuration record created by clearmake or clearaudit

SYNOPSIS

catcr [**–r·ecurse** | **–fla·t** | **–uni·on** | **–che·ck** [**–uni·on**] | **–mak·efile**]
[**–sel·ect** *do-leaf-pattern*] [**–ci**] [**–ele·ment_only**] [**–vie·w_only**]
[**–typ·e** { **f** | **d** | **l** } …] [**–nam·e** *tail-pattern*] [**–zer·o**] [**–wd**]
[**–nxn·ame**] [**–l·ong** | **–s·hort**] *do-pname* …

DESCRIPTION

Displays the *configuration records* (CRs) for the specified *derived objects* (DOs) and, optionally, for their build dependencies. *clearmake* creates a CR each time it executes a build script that creates one or more DOs.

See the *config_record* manual page for a detailed description of the contents of a CR, and for a description of *configuration record hierarchies*. See the *derived_object* manual page for a description of derived objects.

Config Recs and clearaudit

The *clearaudit* utility produces a CR when it exits. In this case, the "build" consists of all commands executed in the audited shell.

Controlling the Report

catcr allows precise control over report contents and format. It includes both input and output filters, and supports a variety of report styles. Input filters, such as `–select`, control which DOs are "visited". All visited DOs can potentially appear in the final listing. Output filters, such as `–view_only`, control which DOs actually appear in the final listing. Often, this is a subset of all visited DOs.

You can:

- Generate a separate report for each derived object on the command line (default), or a single, composite report for all derived objects on the command line (`–union`).
- Specify which derived objects should be considered when compiling report output. The `–recurse`, `–flat`, `–ci`, and `–select` options control which subtargets are visited. They generate recursive or flat-recursive reports of subtargets, visit checked-in DOs, and allow you to visit DOs with a particular name only.
- Select the kind(s) of items that will appear in the report: elements only (`–element_only`), view private objects only (`–view_only`), files, directories, or links (`–type`), or names matching a particular tail pattern (`–name`).
- Display the CR in makefile format, rather than the section-oriented format described above (`–makefile`).
- Choose a normal, long, or short report style. Expanding the listing with `–long` adds comments and supplementary information; restricting the listing with `–short` lists file system objects only. You can also list simple pathnames rather than version-extended pathnames (`–nxname`), and relative pathnames rather than full pathnames (`–wd`).

The –check option determines if the CR contains any "unusual" entries. For example, it determines if the CR contains multiple versions of the same element, or multiple references to the same element with different names.

By default, *catcr* suppresses a CR entirely if the specified filters remove *all* objects (useful for searching). With the –zero option, the listing includes the headers of such CRs.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Reporting on Derived-Object Subtargets. *Default: catcr* lists the derived-object subtargets used to build *do-pname*, but it does not examine or display subtarget CRs. The –recurse, –flat, –union, –check, and –makefile options all direct *catcr* to recurse into subtarget CRs. Use –select to isolate the CRs of one or more subtargets; use –ci to examine the CRs of pre-built, checked-in *DO versions*.

–r·ecurse Displays the CRs of any derived objects that are subtargets of *do-pname*. Each CR is displayed separately.

–fla·t Similar to –recurse, but consolidates the CRs into a single list of versions and derived objects, with no duplicate entries. –flat produces one report for each *do-pname* on the command line. The report includes file system objects only; no headers, variables and options, or build scripts. A number preceding each file name indicates the total number of times it was referenced during the build.

–uni·on Produces *one* report for all derived objects on the command line. Like –flat, it consolidates the CRs of each *do-pname* and its subtargets into a single list of objects, with no duplicate entries. It then combines the separate lists into a single report with no duplicates. The report includes file system objects only; no headers, variables and options, or build scripts.

–che·ck [**–uni·on**]
Flags entries in the CR that have unusual characteristics. It may optionally be specified with –union. This option determines if a CR contains:

- Versions which are not currently checked in — This includes versions that no longer exist (an intermediate version that only existed as a private file, for example), versions that are currently checked out, and versions that were explicitly removed with the *rmver* command.
- Multiple versions of the same element — This could occur, for example, if a build used multiple libraries, which were built from different source versions.
- Multiple references to the same element with different names, such as a renamed element in different directory versions.

–mak·efile Similar to –recurse, but displays the CR in simple *makefile* format. The listing includes the dependencies and build script for each of the derived object's subtargets. You should always include the –wd option with –makefile; this causes *catcr* to list pathnames with respect to the initial working directory of the build. (Note that this differs from the standard behavior of –wd; see below). If you fail to include –wd, *cleartool* displays a warning message, and then

displays the makefile without modifying dependency pathnames.

–sel·ect *do-leaf-pattern*
Starts the listing at the subtarget(s) of *do-pname* that match the specified pattern. *do-leaf-pattern* can be a pattern (see the *wildcards_ccase* manual page) that matches a simple file name — it must not include a slash character (`/`) or the ellipsis wildcard (…). Alternatively, it can be a standard pathname of a derived object.

This option is useful for isolating a derived object that was built as a dependency of another one. For example, this command displays the CR of the derived object named *hello.o* that was used to build *hello* in the current view:

```
% cleartool catcr –select hello.o hello
```

–ci (for use in recursive listings only)
By default, recursive listings stop at *DO versions* — DOs that have been checked in as versions of elements, and used as sources during the build. This option allows you to recurse into the CRs of DO versions. `–ci` only has effect with `–recurse`, `–flat`, and `–union`.

Specifying Kinds of Objects to Display. *Default: catcr* reports on all objects in the CR, which may include: source files, directories, and symbolic links; derived objects; makefiles; view-private files, and non-MVFS objects that were explicitly declared as dependencies.

–ele·ment_only
Lists versions of elements only, including checked-out versions. This option excludes from the listing derived objects (except DO versions), view-private files and directories, symbolic links, and non-MVFS objects.

–vie·w_only
Lists view-private objects only, including checked-out versions of elements. If you specify this option along with `–element_only`, the listing includes just checked-out versions of elements.

–typ·e { **f** | **d** | **l** } …
Restricts the listing to files only (`f`), or to directories only (`d`), or to links only (`l`). If you omit this option: a `–short` listing includes files only; a `–long` listing includes all three kinds. To specify multiple kinds of objects, group them into a single argument: `-type fd`.

–nam·e *tail-pattern*
Restricts the "MVFS objects" listing to those whose final pathname component(s) match the specified pattern. *tail–pattern* can include any of the wildcard characters described in the *wildcards_ccase* manual page.

Controlling Report Appearance. *Default: catcr* reports, in three sections, on MVFS objects, variables and options, and the build script. The report uses full pathnames, and it omits comments and directory versions.

–l·ong Expands the listing to include the kinds of objects in the CR, and comments. With `–makefile`, adds comments only. For example, an object may be listed as a *version*, a *directory version*, or *derived object* (see `ls –long` for a complete list). Comments indicate if an object is `in makefile`, a `referenced derived object`, or a `new derived object`.

–s·hort Restricts the listing to file system objects only (omits header information, variables and options, and build scripts). With `–makefile`, the listing also includes build scripts.

–nxn·ame Lists simple pathnames for MVFS objects, rather than version-extended pathnames or DO-IDs.

–wd Prints pathnames relative to the current working directory, rather than full pathnames. With `–makefile`, displays pathnames relative to the initial working directory of the build.

–zer·o Prints the CR header and options section, even if the specified filters remove all objects. The listing will include the target name, current view, and so on, but no information on particular file system objects.

Specifying the Derived Object(s). *Default:* None.

do-pname …

One or more pathnames, specifying the derived objects whose CRs are to be included in the listing. A standard or view-extended pathname specifies the DO in the view. An extended pathname with a *DO–ID* specifies a particular DO, irrespective of view (for example, `hello.o@@24-Mar.11:32.412`).

Use the *lsdo* command to list derived objects with their DO-IDs.

do-pname can be a DO version, specified with any of ClearCase's version-specification methods (standard pathname, version-extended pathname, and so on).

EXAMPLES

NOTE: Most examples show the same CR processed with different options. Some output lines have been split for clarity.

- List the CR for a derived object in the current view named *bgrs*.

```
% cleartool catcr bgrs
Target bgrs built by jones.dvt
Host "oxygen" running SunOS 4.1.1 (sun4c)
Reference Time 11-Dec-92.12:02:39, this audit started 11-Dec-92.12:04:52
View was oxygen:/home/jones/views/920615.vws
Initial working directory was /vobs/docaux/bgr/sun4
----------------------------
MVFS objects:
----------------------------
/vobs/docaux/bgr/libbgr/sun4/libbgr.a@@10-Dec.16:45.1893
/vobs/docaux/bgr/sun4/bgrs@@11-Dec.12:05.1956
/vobs/docaux/bgr/sun4/buga.o@@11-Dec.12:04.1926
  .
  .
  .
/vobs/docaux/bgr/sun4/bugs.o@@11-Dec.12:03.1902
/vobs/docaux/bgr/sun4/bugsched.o@@11-Dec.12:04.1953
  .
  .
  .
----------------------------
Variables and Options:
----------------------------
CC=/usr/bin/cc
CFLAGS=-I../libbgr -DBSD -DSCCS -g
```

```
ENV_LDFLAGS=../libbgr/sun4/libbgr.a
OBJECTS=main.o pick.o bugs.o bugr.o bugi.o bugf.o bugc.o bugl.o buge.o
bugd.o buga.o bugh.o bugw.o bugfld.o bugdt.o bugu1.o bugu2.o bugsched.o
----------------------------
Build Script:
----------------------------
        /usr/bin/cc -I../libbgr -DBSD -DSCCS -g main.o pick.o bugs.o
bugr.o bugi.o bugf.o bugc.o bugl.o buge.o bugd.o buga.o bugh.o bugw.o
bugfld.o bugdt.o bugu1.o bugu2.o bugsched.o
-o bgrs ../libbgr/sun4/libbgr.a
----------------------------
```

- Combine all CRs associated with *bgrs* and its subtargets into a single listing.

  % cleartool catcr –flat bgrs

```
----------------------------
MVFS objects:
----------------------------
    1 /vobs/docaux/bgr/buga.c@@/main/1                    <19-Dec-91.11:49:03>
    1 /vobs/docaux/bgr/bugc.c@@/main/1                    <19-Dec-91.11:49:09>
    1 /vobs/docaux/bgr/bugd.c@@/main/1                    <19-Dec-91.11:49:14>
   20 /vobs/docaux/bgr/bugs.h@@/main/3                    <17-Jun-92.23:55:22>
    1 /vobs/docaux/bgr/bugsched.c@@/main/1                <19-Dec-91.11:50:07>
  .
  .
  .
    2 /vobs/docaux/bgr/sun4/bugw.o@@11-Dec.12:04.1932
    2 /vobs/docaux/bgr/sun4/main.o@@11-Dec.12:03.1896
```

  The integer at the beginning of an entry indicates the number of times the object was referenced during the build. For example, */vobs/docaux/bgr/bugs.h* was referenced 20 times.

- Excerpt from the CR for the *bugsched.o* subtarget of *bgrs* the versions of elements involved in the build.

  % cleartool catcr –select bugsched.o –element_only bgrs

```
Target bugsched.o built by akp.user
Host "oxygen" running SunOS 4.1.1 (sun4c)
Reference Time 11-Dec-92.15:23:21, this audit started 11-Dec-92.15:23:39
View was neptune:/usr/people/akp/views/920615.vws
Initial working directory was /vobs/docaux/bgr/sun4
----------------------------
MVFS objects:
----------------------------
/vobs/docaux/bgr/bugs.h@@/main/3                    <17-Jun-92.23:55:22>
/vobs/docaux/bgr/bugsched.c@@/main/2                <11-Dec-92.15:23:04>
/vobs/docaux/bgr/libbgr/stint.h@@/main/2            <08-Sep-92.10:06:04>
----------------------------
Variables and Options:
----------------------------
CC=/usr/bin/cc
CFLAGS=-I../libbgr -DBSD -DSCCS -g
RM=rm -f
SRC=..
----------------------------
Build Script:
----------------------------
   rm -f bugsched.o ; /usr/bin/cc -c -I../libbgr -DBSD -DSCCS -g ../bugsched.c
----------------------------
```

- Display in makefile format the CR for a derived object built at pathname *bugi.o*, specified by its DO-ID.

```
% cleartool catcr –wd –makefile bugi.o@@24–Nov.21:45.1623
# Makefile generated 24-Nov-93.21:46:41

# Target bugi.o built by akp.user
# Host "neon" running SunOS 4.1.3 (sun4c)
# Reference Time 24-Nov-93.21:45:19, this audit started 24-Nov-93.21:45:26
# View was neptune:/usr/people/akp/views/930825.vws
# Initial working directory was /vobs/docaux/bgr/sun4
bugi.o: \
  ../bugi.c \
  ../bugs.h \
  ../libbgr/stint.h
        rm -f bugi.o ; /usr/bin/cc -c -I../libbgr -DBSD -DSCCS -g ../bugi.c
```

- List only header files (`.h` suffix) involved in the build of a particular derived object.

```
% cleartool catcr –name '*.h' bgrs
----------------------------
MVFS objects:
----------------------------
   20 /vobs/docaux/bgr/bugs.h@@/main/3              <17-Jun-92.23:55:22>
   19 /vobs/docaux/bgr/libbgr/intstint.h@@/main/1   <19-Dec-91.11:54:50>
   36 /vobs/docaux/bgr/libbgr/stint.h@@/main/2      <08-Sep-92.10:06:04>
    1 /vobs/docaux/bgr/spar.h@@/main/1              <19-Dec-91.11:50:42>
```

SEE ALSO

cleartool subcommands: diffcr, ls, lsdo, rmdo
clearaudit, clearmake, config_spec, config_record, derived_object, wildcards_ccase

NAME catcs – display config spec of a view

SYNOPSIS

catcs [**–tag** *view-tag*]

DESCRIPTION

Displays the contents of a view's *config spec*.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the View. *Default:* Displays the config spec of the current view.

–tag *view-tag*

The view-tag of any view. If the *working directory view* differs from the *set view*, *edcs* displays a warning message and uses the working directory view.

EXAMPLES

- Display the current view's config spec.

```
% cleartool catcs
element * CHECKEDOUT
element * /main/LATEST
```

- Display the config spec of the view with view-tag *jackson_fix*.

```
% cleartool catcs –tag jackson_fix
element * CHECKEDOUT
element * .../rel2_bugfix/LATEST
element -file * REL2 -mkbranch rel2_bugfix
element * /main/LATEST
```

SEE ALSO

cleartool subcommands: edcs, lsview, mktag, pwv, setcs
config_spec

NAME cd – change current working directory

SYNOPSIS

cd [*dir-pname*]

DESCRIPTION

Changes the *current working directory*, just like the standard *cd*(1) command. This command is intended for use in interactive *cleartool* sessions, and in shell scripts that simulate interactive sessions.

Using Extended Pathnames

With a view-extended pathname, *cd* also changes your *working directory view*. The specified view's config spec determines which versions of elements are visible in your new working directory.

With a VOB-extended pathname that specifies an element or branch, *cd* changes your current working directory to a location in *version-extended namespace*, wherein element names and branch names are treated like directories in a read-only file system. The best way to leave version-extended namespace is to *cd* to a full pathname. `cd ..` commands do not exit version-extended namespace until you "ascend" past the VOB root directory. (See the *pathnames_ccase* manual page.)

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the New Working Directory. *Default:* Changes to your home directory, determined by examining the password database.

dir-pname The pathname of the directory to become your current working directory. You can specify a view-extended or version-extended pathname, as described above.

EXAMPLES

- Change to the *release* subdirectory of the current working directory's parent.

```
cleartool> cd ../release
element * CHECKEDOUT
element * .../rel2_bugfix/LATEST
element -file * REL2 -mkbranch rel2_bugfix
element * /main/LATEST
```

- Change to your home directory.

```
cleartool> cd
element * CHECKEDOUT
element * .../rel2_bugfix/LATEST
element -file * REL2 -mkbranch rel2_bugfix
element * /main/LATEST
```

- Use a view-extended pathname to change to the *src* directory in the context of the *jackson_old* view.

```
cleartool> cd /view/jackson_old/usr/hw/src
element * CHECKEDOUT
element * .../rel2_bugfix/LATEST
element -file * REL2 -mkbranch rel2_bugfix
element * /main/LATEST
```

- Change to the directory in extended namespace that represents the *main* branch of element *hello.c*.

```
cleartool> cd hello.c@@/main
element * CHECKEDOUT
element * .../rel2_bugfix/LATEST
element -file * REL2 -mkbranch rel2_bugfix
element * /main/LATEST
```

- Change to a directory in extended namespace, and then return to the original directory.

```
cleartool> cd src@@
cleartool> pwd
/view/jackson_vu@@/usr/hw/main/2/src
cleartool> cd /usr/hw/src
cleartool> pwd
/usr/hw/src
```

SEE ALSO

cleartool subcommands: pwd, pwv
config_spec, pathnames_ccase, cd(1)

NAME checkin – create permanent new version of an element

SYNOPSIS

checkin | **ci** [**–kee·p** | **–rm**] [**–fro·m** *source-pname*] [**–cr**] [**–pti·me**]
[**–ide·ntical** [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* …

DESCRIPTION

For one or more elements, creates a *successor* to a version that was previously checked out in the current view — the *predecessor* version. The *version number* of the successor is one greater than that of the predecessor (except where *rmver* has been used to delete one or more versions from the end of the branch). An appropriate message is displayed:

```
Checked in "msg.c" version "/main/motif/26".
```

A checkin record is created, which can be listed with the *lshistory* command:

```
% cleartool lshistory msg.c
06-Aug.12:09   akp    create version "msg.c@@/main/motif/26"
  .
  .
  .
```

Only *elements* can be checked in. You cannot simply checkin a view-private file, but must first make an element of the same name. Use the `mkelem -ci` command to simultaneously create an element and checkin a view-private file as its first version.

By default, the new version of a file element is created by copying the contents of the view-private file named *pname* (the *checked-out version*), and then deleting that file. The `–keep` and `–from` options alter this behavior.

After the element is checked in, your view typically selects the version you just created. But it is possible that your view will select another version (perhaps on another branch). In this case, *checkin* displays a warning message.

CHECKIN OF RESERVED AND UNRESERVED CHECKOUTS

At the time you enter a *checkin* command, there may be several checkouts of the same version. At most one of the checkouts (perhaps yours) is *reserved* — all the others are *unreserved*. Your *checkin* command succeeds in either of these cases:

- yours was a reserved checkout
- all checkouts were unreserved, and no one has checked in a successor version

If the command fails because someone else has a reserved checkout, you must wait until that checkout is resolved, with *checkin*, *uncheckout*, or *unreserve*. If the command fails because someone has checked in a successor version ahead of you, you can checkin your work now, by:

- performing a merge from the current *LATEST* version on the branch to your checked-out version
- entering the *checkin* command again

CHECKIN OF DERIVED OBJECTS

You can checkin a *derived object* to make it a version of an element (a *DO version*). By default, both the data and *configuration record* of a derived object are checked in. To save disk storage, you can use the `–cr` option to checkin *only* the configuration record, not the data.

clearmake can reuse or *wink-in* a derived object only if it is stored under its original pathname. Thus, a DO version created under an alternate name with `checkin -from` cannot be used by *clearmake* for build avoidance. (It can still use the derived object named in the `-from` option, which is unaffected by this command.)

See the *derived_object* manual page for information regarding subsequent operations on DO versions.

PERMISSIONS AND LOCKS

checkin can perform up to three permission checks:

- If the element's *set-UID* bit is set, only the element's owner, the VOB owner, or the *root* user can perform a checkin.
- If the element's *set-GID* bit is set, only a member of the element's group, the VOB owner, or the *root* user can perform a checkout.
- For all elements, an error occurs if you are not the user who checked out the element, the element's owner, the VOB owner, or the *root* user.

Even if you have permission to execute this command, it fails if any of the following objects have been locked: VOB, element type, branch type, element, branch, pool (file elements only).

OPTIONS AND ARGUMENTS

Managing Source Files. *Default: checkin* deletes each view-private, checked-out *pname* file after using it to create a new version. You can use the following three options (which have no meaning for directory elements) to save view-private copies, or to check in source files from other locations.

−kee·p Saves the current contents of each checked-out version in a view-private file, in addition to creating a new version. The view-private file gets a name of the form *pname**.keep*** (or possibly, *pname**.keep.**n*). `-keep` is the default when you use the `-from` option, since the current contents of the checked-out version would otherwise be lost.

−rm Removes each view-private *pname* file after creating a new version. This is the default if you do not use the `-from` option.

−fro·m *source-pname*
Uses the contents of *source-pname* as the new version, instead of the view-private file *pname*. By default, `-keep` is invoked to preserve the contents of the view-private *pname*. The *source-pname* file itself is unaffected. This option makes it easy to copy data from another location (outside the VOB, perhaps) into an element's version tree.

When using this option, specify only one *pname* argument.

Checking In Derived Objects. *Default: checkin* checks in both the data and configuration record for a derived object.

−cr (for derived-object checkin only)
Checks in only the *configuration record* for the specified derived object(s). Each new DO version will have a configuration record, but no data. You can use many *cleartool* commands with such DO versions, such as *catcr*, *diffcr*, and *mklabel* (but not *lsdo*). It is also visible to standard UNIX *ls*(1). However, a version created with this option cannot be opened or executed, since there is no data.

Miscellaneous Options. *Default: checkin* resets the new version's modification time to the checkin time. Also, *checkin* cancels the checkin operation for some types of *pname* files, if their contents match their predecessor versions.

–pti·me Preserves the modification time of the file being checked in. If you omit this option, *cleartool* sets the modification time of the new version to the checkin time.

NOTE: On some platforms, it is important that the modification time be preserved for archive files (libraries) created by *ar*(1) (and perhaps updated with *ranlib*(1)). The link editor, *ld*(1), will complain if the modification time does not match a time recorded in the archive itself. Be sure to use this option, or (more reliably) store archive files as elements of a user-defined type, created with the `mkeltype -ptime` command. This causes `–ptime` to be invoked automatically when the element is checked in.

–ide·ntical Checks in the element even if the predecessor version is identical to the checked-out version. By default, the checkin operation is cancelled in such cases.

NOTE: This situation applies only to elements whose *type manager* computes version-to-version *deltas* (for example, elements of type *text_file* and *compressed_text_file*). If an element's type manager does not compute deltas, *checkin* always creates a new version, whether or not it is identical to its predecessor. For example, a new version will always be created for an element of type *file*, which uses the *whole_copy* type manager.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

NOTE: If a checkout comment exists (specified with `checkout -c` and/or automatically generated to record changes to a checked-out directory), you can make it the *checkin* comment:

- Use `checkin -nc`.
- Use `checkin -cqe` and type `<Ctrl-D>` or `.<Return>` at the prompt.

Any other entry at the `–cqe` prompt specifies a new checkin comment, discarding the checkout comment (if any) for that element. The `–c` and `–cq` options always discard the checkout comment (if any) for each element processed.

Element Argument(s). *Default:* None.

pname … The pathnames of one or more elements to be checked in.

EXAMPLES

- After verifying its checkout comment, checkin element *util.c*, using that comment.

```
% cleartool lscheckout –long util.c
10-Dec-92.16:11:07   Chuck Jackson (jackson.dvt@oxygen)
  checkout version "util.c" from /main/4 (reserved)
  by view: "oxygen"/home/jackson/cj.vws"
  "revise syntax"
```

```
% cleartool checkin –nc util.c
Checked in "util.c" version "/main/5".
```

- Checkin an element from an alternate file, discarding the checked-out version. Provide a comment on the command line.

```
% cleartool checkin –rm –from /usr/tmp/util.c –c "Release 1.1 update" util.c
Checked in "util.c" version "/main/6".
```

- Checkin only the configuration record of a derived object, discarding its data.

```
% cleartool checkin –nc –cr hello
Checked in "hello" version "/main/1".
```

- Checkin all elements in the current VOB that are checked-out to the current view. Specify a single comment for all the `create version` event records.

```
% cleartool checkin –cq ‘cleartool lscheckout –all –me –cview –short‘
Comment for all listed objects:
checkpoint before vacation
.
Checked in "/usr/hw/src/hello.h" version "/main/2".
Checked in "/usr/hw/release" version "/main/1".
Checked in "/usr/hw/src/util.h" version "/main/1".
  ⋮
```

SEE ALSO

cleartool subcommands: checkout, lshistory, merge, mkelem, mkeltype, rmver, uncheckout
clearmake, derived_object, profile_ccase, touch(1)

NAME checkout – create view-private, modifiable copy of a version

SYNOPSIS

checkout | **co** [**–res·erved** | **–unr·eserved**] [**–bra·nch** *branch-pname*]
[**–out** *dest-pname* | **–nda·ta**]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* …

DESCRIPTION

For one or more elements, checks out a branch (or equivalently, the most recent version on a branch). In most cases, this creates a writable copy of that version in the current view (the *checked-out version*), but see "Checking Out a DO Version" below. An appropriate message is displayed — for example:

```
Checked out "msg.c" from version "/main/motif/25".
```

A checkout record is created, which can be listed with the *lscheckout* command:

```
% cleartool lsco msg.c
05-Aug.20:50   akp    checkout version "msg.c" from /main/motif/25 (reserved)
```

You can checkout only the most recent version on a branch. To modify an earlier version, you must create a subbranch at that version. (See the *mkbranch* manual page). Furthermore, from a single view, you can checkout only one branch of an element at a time.

You can checkout a version that your view does not currently select, either by using the `–branch` option or by specifying a *pname* argument that includes a branch pathname (for example, *msg.c@@/main/rel4_bugfix*). In such cases, a warning message appears:

```
cleartool: Warning: Version checked out is different from
version previously selected by view.
```

If a view-private object already exists with the same name as an element being checked out, *checkout* saves the private object as *pname**.keep*** (or possibly, *pname**.keep**.n*).

Before using a command that changes the contents of a directory (*mkelem*, *mkdir*, *rmname*, *ln*, or *mv*), you must first checkout the directory. Each of these commands automatically appends an appropriate line to the directory's checkout comment. For example, using *mkelem* to create a new element within a directory might add this comment line:

```
Added file element "wel.c".
```

RESERVED AND UNRESERVED CHECKOUTS

A version can have at most one *reserved checkout* and any number of *unreserved checkouts*. Performing a reserved checkout guarantees you the right to create a *successor* to the version you checked out. If several users perform unreserved checkouts, any one of them (and just one) can create a successor version. Each other user with an unreserved checkout must perform a merge before checking his or her versions back in as a further successor.

Each reserved or unreserved checkout of a version must be performed in a different view.

You can change the checkout status of a checked-out version with the *reserve* and *unreserve* commands.

CHECKING OUT A DO VERSION

If the version being checked out is a derived object (*DO version*), *checkout* attempts to *wink-in* the DO to your view. If it cannot perform the wink-in, it copies the DO's data instead. A wink-in cannot be performed if you use the `–out` option to specify a destination in another VOB, or in a non-VOB location, such as */tmp*.

See the *derived_object* manual page for additional information on the behavior of checked-out DO versions.

AUTO-MAKE-BRANCH

If the view selects a version using a config spec rule with a **–mkbranch** *branch-type* clause:

1. *checkout* first creates a branch of type *branch-type*.
2. It checks out (version 0 on) the newly-created branch.

Except for some extra messages, this appears little different from an ordinary checkout. The checked-out version has the expected contents, because version 0 on the new branch has the same contents as the version at the branch point.

Multiple-Level Auto-Make-Branch

A config spec can include a "cascade" of *auto-make-branch* rules, causing *checkout* to create multiple branching levels at once. *checkout* keeps performing auto-make-branch until version 0 on the newly-created branch is *not* selected by a rule with a `–mkbranch` clause. For example:

```
element * CHECKEDOUT                              (1)
element * .../br2/LATEST                          (2)
element * .../br1/LATEST -mkbranch br2            (3)
element * MYLABEL -mkbranch br1                   (4)
element * /main/LATEST                            (5)
```

If you checkout an element in a view that currently selects the version labeled *MYLABEL*:

1. A branch of type *br1* is created at the *MYLABEL* version (Rule 4).
2. Rule 3 now selects the newly-created version *.../br1/0*, so a branch of type *br2* is created at that version.
3. Version *.../br1/br2/0* is checked out. The checked-out version has the same contents as the *MYLABEL* version, and is selected by Rule 1. When you edit and *checkin* a new version, *.../br1/br2/1*, the view will select it with Rule 2.

CHECKED-OUT FILES

A checked-out file is a view-private object, which can be read, edited, and even deleted like any standard file. The initial permissions on the checked-out file are determined by this algorithm:

- start with the permissions of the element itself (see the *mkelem* and *protect* manual pages)
- add a "write" permission wherever the element itself has a "read" permission (user, group, and/or other)
- subtract "read", "write", and/or "execute" permissions according to your current *umask*(1) value

You can change the permissions of the checked-out file with the standard UNIX *chmod*(1) command. But you must use the ClearCase *protect* command to change the permissions of the element itself.

PERMISSIONS AND LOCKS

checkout can perform up to three permission checks:

- If the element's *set-UID* bit is set, only the element's owner, the VOB owner, or the *root* user can perform a checkout.
- If the element's *set-GID* bit is set, only a member of the element's group, the VOB owner, or the *root* user can perform a checkout.
- For all elements, an error occurs if you are not a member of the element's group, the element's owner, the VOB owner, or the *root* user.

Even if you have permission to execute this command, it fails if any of the following objects have been locked: VOB, element type, branch type, element, branch.

OPTIONS AND ARGUMENTS

Reserved and Unreserved Checkouts. *Default: checkout* reserves the branch.

−res·erved "Reserves" the branch: no user in another view can perform a reserved checkout of the same branch (but any number of unreserved checkouts can be performed); no new versions can be created on the branch until your checkout is resolved with *checkin* or *uncheckout*.

−unr·eserved
Leaves the branch *unreserved*: other users, in other views, can checkout the same version (but at most one of the checkouts can be reserved).

See the *checkin* manual page for a discussion of how new versions are created from reserved and unreserved checkouts.

Non-Standard Checkouts. *Default:* If *pname* specifies a particular branch, checkout that branch — that is, the latest version on the branch. Otherwise, checkout the branch containing the version specified by *pname*. *checkout* creates a view-private copy of each checked-out version and names it *pname* in view-private storage.

−bra·nch *branch-pname*
Specifies the branch whose most recent version is to be checked out. If you omit this option, *cleartool* determines the branch from the *pname* argument, or uses the branch whose version is currently selected by the view.

Creation of Checked-Out Version in View. *Default:* (file elements only) Copies the version being checked out (that is, the most recent version on the branch being checked out) to a view-private file with the same pathname as the element; exception: if the version being checked out is a derived object, it is winked-in to the view.

−out *dest-pname*
(does not apply to directories or DO versions) Creates a writable file under an alternate file name (perhaps in a different directory). No view-private file named *pname* is created. The *ls* command lists the element as `checkedout but removed`.

–nda·ta (does not apply to directories) Creates a checkout record for the version, but does not create an editable file containing its data. The *ls* command lists the file as `checkedout but removed`.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Element Argument(s). *Default:* None.

pname … Pathnames of one or more elements to be checked out. By default, the branch whose version is currently selected by the view is checked out. You can override this by specifying a `–branch` option, or by specifying the version-extended pathname of a branch or version. (If you specify an "old" version, *cleartool* still checks out the most recent version on its branch.)

EXAMPLES

- Checkout the currently-selected version of element *hello.c*, with no comment.

```
% cleartool checkout –nc hello.c
Checked out "hello.c" from version "/main/3".
```

- Checkout the latest version on the *rel2_bugfix* branch of file *msg.c*, to an alternate file name.

```
% cleartool checkout –nc –branch /main/rel2_bugfix –out msg_test.c msg.c
Checked out "msg.c" from version "/main/rel2_bugfix/1".
% cleartool ls
  .
  .
  .
msg_test.c
msg.c@@/main/rel2_bugfix/CHECKEDOUT from /main/rel2_bugfix/1 [checkedout but removed]
  .
  .
  .
```

- Checkout the latest version on the *rel2_bugfix* branch of file *msg.c*, using an extended pathname to indicate the branch.

```
% cleartool checkout –nc msg.c@@/main/rel2_bugfix
Checked out "msg.c" from version "/main/rel2_bugfix/1".
```

This command checks out the same version as the preceding example.

- Perform an unreserved checkout of element *hello.h*. Provide a comment on the command line.

```
% cleartool checkout –c "modify local defines" –unreserved hello.h
Checked out "hello.h" from version "/main/2".
```

- Checkout *hello.c*. Then, change your mind and cancel the checkout, removing the view-private copy.

```
% cleartool checkout –nc hello.c
Checked out "hello.c" from version "/main/1".
% cleartool unco –rm hello.c
Checkout cancelled for "hello.c".
```

SEE ALSO

cleartool subcommands: checkin, lscheckout, ls, mkelem, mkbranch, protect, reserve, uncheckout, unreserve clearmake, config_spec, derived_object, profile_ccase

NAME chevent – modify comment string in existing event record

SYNOPSIS

- Modify event records of objects by specifying names:

chevent [**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–app·end** | **–ins·ert** | **–rep·lace**]
{
pname …
| { **–elt·ype** | **–brt·ype** | **–att·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–rpt·ype** }
[**–vob** *pname-in-vob*] *type-name …*
| **–poo·l** [**–vob** *pname-in-vob*] *pool-name …*
| **–hli·nk** *hlink-selector …*
| **–vob** *pname-in-vob*
| **–vre·plica** [**–vob·pname-in-vob**] *replica-name …*
}

- Modify event records of objects by specifying event-IDs:

chevent [**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–app·end** | **–ins·ert** | **–rep·lace**]
–eve·nt [**–vob** *pname-in-vob*] *event-ID …*

DESCRIPTION

Modifies or replaces the comment string in one or more existing *event records*. This command is useful for correcting typing errors, and for including information that was omitted in the original comment.

There are several ways to specify an event record whose comment you want to change:

- If you specify a checked-out version, *chevent* changes the comment in the checkout event record.
- If you specify any other object, *chevent* changes that object's creation event record. For example, if you specify a label type object, *chevent* changes the comment supplied when that label type was created with *mklbtype*.
- You can change the comment in an arbitrary event record by passing its *event-ID* to the `–event` option. Use the command `lshistory -eventid` to capture event-IDs. (Event IDs remain valid until the VOB is reformatted with *reformatvob*.)

See the *events_ccase* manual page for details on the operations that cause event records to be created, and how event records are attached to objects. See also "Event Records and Comments" in the *cleartool* manual page.

PERMISSIONS AND LOCKS

Permissions Checking: To modify an event's comment, you must be the user associated with the event, the VOB owner, or the root user. *Locks:* Even if you have permission to execute this command, locks cause it to fail as follows:

Object	Locks that Prevent Changing the Object's Events
VOB	VOB
pool	VOB, pool
type	VOB, type
symbolic link	VOB
element	VOB, element type, element
branch, version	VOB, element type, element, branch type, branch
hyperlink	VOB, hyperlink type

See also "Permissions Checking" in the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Comment Change. *Default:* For each object or event, *chevent* prompts for a comment string to apply to the corresponding event record.

–c *comment* Specifies a character string to replace the existing comment or be added to it.

–cq Prompts for one comment, which will be used to update all of the event records.

–cqe Same as default — prompts for a separate comment string for each object or event ID.

–nc No comment. When combined with `–replace`, this option removes the existing comment. Otherwise, it nullifies the effect of *chevent*.

Specifying How to Change the Comment. *Default:* The new comment is appended to the existing one, separated by a `<NL>` character.

–app·end Same as default.

–ins·ert The new comment is inserted before the existing one, followed by a `<NL>` character.

–rep·lace The existing comment is discarded; the new comment replaces it.

Specifying Event Records to Be Changed. *Default:* None.

pname …
{ **–elt·ype** | **–brt·ype** | **–att·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–rpt·ype** } [**–vob** *pname-in-vob*] *type-name*
–poo·l [**–vob** *pname-in-vob*] *pool-name …*
–hli·nk *hlink-selector …*
–vob *pname-in-vob …*
–vre·plica [**–vob** *pname-in-vob*] *replica-name …*
–eve·nt [**–vob** *pname-in-vob*] *event-ID …*

Names of one or more objects, or (with `–event`) one or more numerical *event-IDs*. Use the same syntax as with *lock* to specify an object. Specifying a checked–out version changes its `checkout version` comment. You can use any of the following to specify the checked-out version:

`hello.h`	*(standard pathname)*
`hello.h@@/main/rel2_bugfix/CHECKEDOUT`	*(extended pathname to checked–out 'placeholder' version)*

`hello.h@@/main/rel2_bugfix/CHECKEDOUT.465` *('placeholder' version has unique numeric suffix)*

You must specify a VOB object using its VOB-tag or a pathname thereunder. (You cannot use the pathname of the VOB storage directory.) The *pname* argument(s) must be specified after all options; the `–event` keyword can appear anywhere an option is valid.

To determine the event-ID of an event, use `lshistory -eventid`.

Specifying the VOB. *Default:* Use the VOB containing the current working directory.

–vob *pname-in-vob*

With `–xxtype`, `–pool`, `–hlink`, `–event`, or `–vreplica`, this option specifies the VOB in which the event record(s) are to be changed.

You can also use an option in this form to indicate a VOB object whose creation event record is to be changed.

The *pname-in-vob* can be any location within the VOB.

EXAMPLES

- Add a creation comment for an element:

```
% cleartool chevent hello.c@@
Comments for "hello.c":
Main module of greeting program.
.
Modified event of file element "hello.c".
% cleartool describe hello.c@@
file element "hello.c@@"
  created 04-Dec-93.14:38:26 by anne.user
  "Main module of greeting program."
  element type: text_file
  source pool: p1  cleartext pool: pc1
```

- Add a header to a checked-out version's checkout comment:

```
% cleartool lscheckout bye.c
13-May.13:58  anne    checkout version "bye.c" from /main/11 (reserved)
  "Improve error handling."
% cleartool chevent –insert –c "Fix bug #2493:" bye.c
Modified event of version "bye.c".
% cleartool lscheckout bye.c
13-May.13:58  anne    checkout version "bye.c" from /main/11 (reserved)
  "Fix bug #2493:
   Improve error handling."
```

- Update a label type creation comment:

```
% cleartool chevent –append –brtype v1_bugfix
Comments for "v1_bugfix":
Branches should sprout from the version labeled "V1"
.
Modified event of branch type "v1_bugfix".
% cleartool lstype –brtype v1_bugfix
28-Mar.16:26   ali     branch type "v1_bugfix"
```

```
  "Branch for fixes to version 1.
   Branches should sprout from the version labeled "V1""
```

- Delete the comment on a branch object:

```
% cleartool chevent –replace –nc welcome.c@@/main/v1_bugfix
Modified event of branch "welcome.c".
```

- Find the event ID for an operation and append a comment string to the one already assigned to that event. Then check that the new comment was added.

```
% cleartool lshistory –long –eventid util.c
event 45678:
21-Mar-94.14:45:20     Anne Duvo   (anne@neptune)
  destroy sub-branch "bugfix" of branch "util.c@@/main"
  "Destroyed branch "/main/bugfix"."
  .
  .
  .
```

```
% cleartool chevent –c "bugfix merge completed." –append –event 45678
Modified event "45678".
```

```
% cleartool lshistory –long –eventid util.c
event 45678:
21-Mar-94.14:45:20     Anne Duvo   (anne@neptune)
  destroy sub-branch "bugfix" of branch "util.c@@/main"
  "Destroyed branch "/main/bugfix".
  "bugfix merge completed."
  .
  .
  .
```

SEE ALSO

cleartool subcommands: lshistory, lock, mktrtype
vob_scrubber, events_ccase

NAME chpool – change the storage pool to which an element is assigned

SYNOPSIS

chpool [**–for·ce**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pool-name pname …*

DESCRIPTION

Changes the *source storage pool*, *derived object storage pool*, or *cleartext storage pool* to which one or more elements are assigned.

For a file element:

- Changing the source pool moves the data container(s) that store all existing versions from the current pool to the specified pool.
- Changing the cleartext pool designates a different location for new *cleartext* versions. Existing cleartext versions remain where they are, and will eventually be scrubbed. (See the *scrubber* manual page.)
- An error occurs if you attempt to assign a file element to a derived object pool; file elements have source and cleartext pools only.

For a directory element:

- Changing the source pool or the cleartext pool affects *pool inheritance* by new elements: in the future, elements created within the directory will be assigned to the new pool. The pool assignments of existing elements remain unchanged.
- Changing the derived object pool designates a new location for *shared* derived objects with pathnames in that directory: in the future, the *promote_server* program will copy data containers to the new pool. The existing contents of the old pool remain unchanged, and will eventually be deleted by *scrubber*.

Commands for Listing Pools

The *lspool* command lists a VOB's storage pools. The *describe* command includes storage pool assignments in its listing for an element — to reference an element (rather than one of its versions), append the extended naming symbol to the element's standard pathname:

```
cleartool describe msg.c@@
```

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, root user.
Locks: An error occurs if any of the following objects are locked: VOB, element type, element, pool. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

User Interaction. *Default:* Prompts for confirmation before moving data containers.

–for·ce Suppresses the confirmation step.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Pool. *Default:* None.

pool-name An existing storage pool.

Specifying the Elements. *Default:* None.

pname … One or more pathnames, each of which specifies a file or directory element. A standard pathname is valid — you need not append the extended naming symbol. (More generally, specifying a version or a branch is equivalent to specifying its element.)

EXAMPLES

- Reassign all elements in the current directory that have a `.c` suffix to cleartext pool *cltxt2*.

```
% cleartool chpool cltxt2 *.c
Changed pool for "cm_add.c" to "cltxt2".
Changed pool for "cm_fill.c" to "cltxt2".
Changed pool for "convolution.c" to "cltxt2".
Changed pool for "msg.c" to "cltxt2".
Changed pool for "test_cmd.c" to "cltxt2".
Changed pool for "util.c" to "cltxt2".
```

- Change the default source pool for the *src* directory, so that new elements created in this directory will be assigned to the `c_pool` pool.

```
% cleartool chpool c_pool src
Changed pool for "src" to "c_pool".
```

- Change the source pool for *hello.c* to *sdft*, the VOB's default source pool. (Assumes the element had been assigned to a different pool.)

```
% cleartool chpool sdft hello.c
Move all versions of element "hello.c"?  [no] yes
Changed pool for "hello.c" to "sdft".
```

SEE ALSO

cleartool subcommands: lspool, mkdir, mkelem, mkpool
promote_server, scrubber, profile_ccase

NAME chtype – change the type of an element / rename a branch

SYNOPSIS

chtype [**−for·ce**] [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *type-name pname* …

DESCRIPTION

Changes the *element type* of one or more existing elements, or renames one or more existing branches. Both these operations involve changing the *type object* associated with the element or branch.

Changing an Element's Type

You can use *chtype* to convert an element from one element type to another (for example, from *file* to *text_file*). Typically, you change an element's type to change the way its versions are stored. For example, versions of a *file* element are stored in separate data containers in a VOB source pool. Converting the element to type *text_file* causes all its versions to be stored in a single data container, as a set of *deltas* (version-to-version differences); this saves disk space.

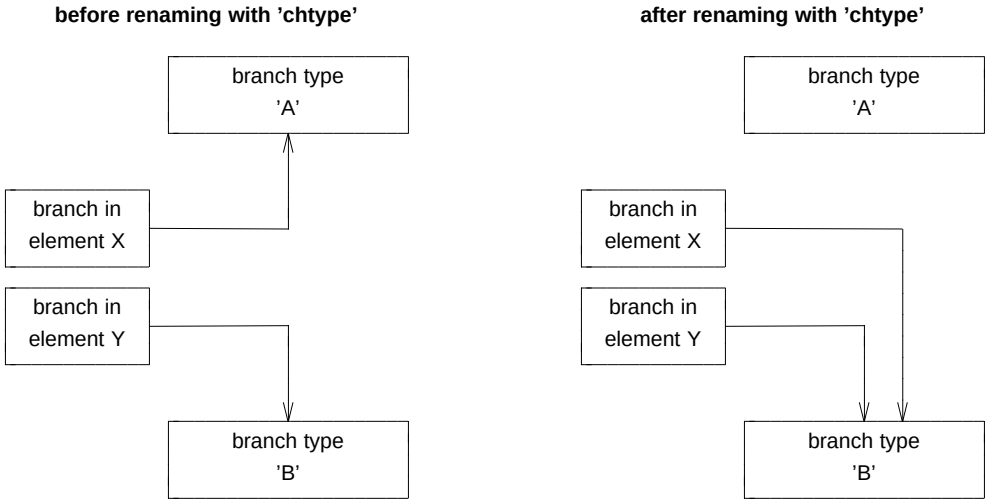
Restrictions. All versions of an element must "fit" the new element type. For example, converting an element to type *text_file* fails if any of its versions contains binary data, rather than ASCII text. You cannot convert files to directories, and vice-versa.

Renaming a Branch

You can use *chtype* to rename a branch (for example, from *bugfix* to *maintenance*). ClearCase implements a branch as an instance of a *branch type* object. Thus, "rename the branch from A to B" actually means "change the branch from an instance of branch type A to an instance of branch type B."

NOTE: Don't confuse the renaming of a particular branch (*chtype*) with the renaming of a branch type (*rntype*). Figure 3 illustrates the difference.

Renaming a Particular Branch:



Renaming a Branch Type:

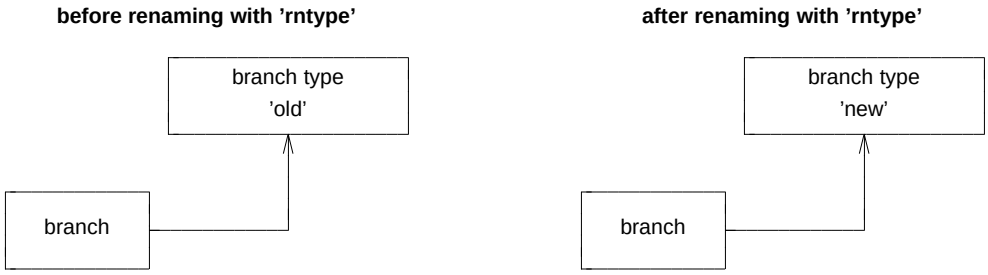


Figure 3. Renaming a Branch vs. Renaming a Branch Type

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: *change element type*: element owner, VOB owner, root user; *change branch type*: branch creator, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: *change element type*: VOB, element type, element, pool (non-directory elements, or new type manager only); *change branch type*: VOB, element type, element, branch type, branch. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Confirmation Step. *Default: chtype* prompts for confirmation if changing an element's type will change the way its versions are stored in the VOB storage pool.

–for·ce Suppresses the confirmation step.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the New Type. *Default:* None.

type-name An existing element type, or an existing branch type.

Specifying the Elements or Branches. *Default:* None.

pname … When changing an element type: One or more pathnames, each of which specifies a file or directory element. A standard pathname is valid — you need not append the extended naming symbol. That is, specifying a version is equivalent to specifying its element. If the *type-name* argument is an element type, supplying a *pname* that terminates with a branch name causes an error.

When renaming a branch: One or more extended pathnames, each of which specifies a particular branch of an element. For example:

```
foo.c@@/main/bugfix
bar.c@@/main/maint/bug405
```

EXAMPLES

- Convert an element to type *file*.

```
% cleartool chtype file hello.c
Change version manager and reconstruct all versions for "hello.c"? [no] yes
Changed type of element "hello.c" to "file".
```

- Rename a branch from *rel2_bugfix* to *maintenance*, providing a comment.

```
% cleartool chtype –c "rel2_bugfix no longer in use" \
     maintenance util.c@@/main/rel2_bugfix
Changed type of branch "util.c@@/main/rel2_bugfix" to "maintenance".
```

- Convert an archive library to *compressed_file* format, suppressing confirmation prompts.

```
% cleartool chtype –force compressed_file libutil.a
Changed type of element "libutil.a" to "compressed_file".
```

SEE ALSO

cleartool subcommands: mkbrtype, mkeltype, mkelem, rntype
cc.magic, profile_ccase

NAME describe – describe an object

SYNOPSIS

- Describe file system objects:

  des·cribe [**–cvi·ew**] [**–l·ong** | **–s·hort** | **–fmt** *format-string*]
  [**–ver·sion** *version-selector* | **–anc·estor**]
  [**–ala·bel** { *label-type-name*[,...] | **–all** }]
  [**–aat·tr** { *attr-type-name*[,...] | **–all** }]
  [**–ahl·ink** { *hlink-type-name*[,...] | **–all** }]
  [**–ihl·ink** { *hlink-type-name*[,...] | **–all** }]
  [**–pre·decessor**] *pname* ...

- Describe type objects:

  des·cribe **–typ·e** [**–l·ong** | **–s·hort** | **–fmt** *format-string*] [**–vob** *pname-in-vob*] *type-name* ...

- Describe derived objects:

  des·cribe *DO-name* ...

- Describe hyperlink objects:

  des·cribe **–hli·nk** *hlink-selector* ...

- Describe storage pools:

  des·cribe **–poo·l** [**–l·ong** | **–s·hort** | **–fmt** *format-string*] [**–vob** *pname-in-vob*] *pool-name* ...

- Describe VOB:

  des·cribe **–vob** *pname-in-vob* [**–l·ong** | **–s·hort** | **–fmt** *format-string*]

- Describe VOB replica:

  des·cribe [**–l·ong** | **–s·hort** | **–fmt** *format-string*] **–vre·plica** [**–vob** *pname-in-vob*] *replica-name* ...

DESCRIPTION

Lists information about VOBs and the objects they contain. For example, you can use *describe* to:

- list the attributes and/or version labels attached to a particular version
- list the hyperlinks attached to a particular object
- list the predecessor of a particular version
- list the views that have checkouts or derived objects in a particular VOB (*describe -long -vob*)

describe produces several kinds of listings:

- **File system data** — Provides information on elements, branches, versions, derived objects, and VOB symbolic links.

  The description of an element (for example, `cleartool describe hello.h@@`) includes a listing of the storage pools to which the element is currently assigned. (See *mkpool* and *chpool* for more information.)

  A version's description includes the *version-ID* of its predecessor version.

  An ordinary derived object is listed with `derived object` in its header; a derived object that has been checked in as a version of an element (*DO version*) is listed with `derived object version` in its header.

- **Type object** — Provides information on a VOB's *type objects* (for example, on a specified list of label types). This form of the command displays the same information as `lstype -long`.
- **Hyperlink object** — Provides information on a hyperlink object.
- **Storage pool** — Provides information on a VOB's source, derived object, and cleartext storage pools. This form of the command displays the same information as `lspool -long`.
- **VOB object or VOB replica** — Provides information on the object that represents the VOB itself, or the object that represents one of its *replicas* (Atria MultiSite product). For a VOB object, this includes such information as its storage area, creation date, owner, and related views.

Unavailable Remote VOB

File system objects can be hyperlinked to objects in another VOB. If the other VOB is currently unavailable (perhaps it has been unmounted), *describe* tries to be helpful:

```
cleartool: Error: Unable to locate versioned object base with object id:
  "51023fa9.68b711cc.b358.08:00:69:02:1d:c7".
  .
  .
  .
Hyperlinks:
  @183@/usr/proj  /usr/proj/elem2.c@@/main/2 -> <object not available>
```

Versions Without Data

The description of a version can include the annotation `[version has no data]`. A file element version can be created without data, using `checkin -cr`; an existing version's data can be removed with `rmver -data`.

This annotation appears for *all* versions of directory elements. This reflects the fact there is no data container in a source storage pool for a directory version — the contents of a directory version are stored in the VOB database.

Hyperlink Inheritance

By default, a version implicitly *inherits* a hyperlink attached to any of its ancestor versions, on the same branch or on a parent branch. Inherited hyperlinks are listed only if you use the `-ihlink` option.

A hyperlink stops being passed down to its descendents if it is superseded by another hyperlink of the same type, explicitly attached to some descendent version. You can use a *null-ended* hyperlink ("from" object, but no "to" object) as the superseding hyperlink to effectively cancel hyperlink inheritance.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Describing Objects in Other Views. *Default:* If you use a view-extended pathname to specify an object in (or as seen through) another view, *describe* lists that view's name for the object:

```
version: "/view/gamma/usr/project/src/util.c"
```

–cvi·ew Lists an object using the current view's name for it:

```
version: "/usr/project/src/all_utils.c"
```

This option is useful when different views select different directory versions, in which elements have been renamed.

Report Format. *Default:* Lists the object's name and some additional information.

–l·ong Expands the listing. With `–vob`, for example, lists all views that have checkouts or derived objects associated with the specified VOB. This listing includes the *UUIDs* of those views, which can be used with *rmview*.

–s·hort Lists only an object's (path)name. The effect is slightly different when used in combination with `–label`, `–attr`, `–hlink`, or `–predecessor`. See the descriptions of these options below.

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

Excerpting Description Information. *Default: describe* lists the predecessor (if the object is a version), and reports on all of the object's version labels, attributes, and hyperlinks. With one or more of the following options, the report includes the requested information only — for example, only a listing of the predecessor version and version label(s).

–ala·bel *label-type-name*[,...]
–aat·tr *attr-type-name*[,...]
–ahl·ink *hlink-type-name*[,...]
–ihl·ink *hlink-type-name*[,...]
–pre·decessor
Specify one or more of these options to excerpt information from the overall description of an object. A list of names of type objects must be comma-separated, with no white space; you can use the special keyword `-all` to specify all types of a particular kind.

If you specify `-short` as well, the listing is restricted even further:

- For `-predecessor`, just the version-ID of the predecessor version is listed.
- For `-alabel`, just the version label(s) are listed.
- For `-aattr`, just the attribute value(s) are listed.
- For `-ahlink`, the listing includes the pathname(s) of the object(s) hyperlinked to *pname*, annotated with `->` (listed object is the "to" object) or `<-` (listed object is the "from" object). For example:

  ```
  -> /usr/proj/include/db.h@@/main/52
  <- /usr/proj/bin/vega@@/main/5
  ```

  Inherited hyperlinks are not included in this listing.
- For `-ihlink`, the listing includes the hyperlinks inherited by *pname*, which must specify a version. Pathnames of both the "from" and "to" objects are listed, one of which is an ancestor of *pname*, or is *pname* itself. (That is, `-ihlink` also includes hyperlinks that are attached to *pname* itself.)

Specifying the Objects to be Described. *Default: describe* expects at least one argument that names an element, branch, version, VOB link, derived object, or hyperlink (*pname*, *DO-name*, or *hlink-selector*). You can use `-version` or `-ancestor` to control the way *pname* arguments are interpreted. To describe a *type* object, *storage pool* object, or the VOB object itself, you must use one of the options `-type`, `-pool`, or `-vob`.

pname … One or more pathnames, indicating objects to be described: elements, branches, versions, or derived objects.

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A standard or view-extended pathname to a derived object specifies the DO in the view.
- An extended pathname specifies an element, branch, version, or derived object, independent of view.

Examples:

`foo.c`	*(version of 'foo.c' selected by current view)*
`foo.o`	*(derived object 'foo.o' built or winked–in to current view)*
`/view/gamma/usr/project/src/foo.c`	*(version of 'foo.c' selected by another view)*
`/view/gamma/usr/project/src/foo.o`	*(derived object 'foo.o' built in another view)*
`foo.c@@/main/5`	*(version 5 on main branch of 'foo.c')*
`foo.o@@11-Nov.09:19.219`	*(derived object, specified by DO–ID)*
`foo.c@@/REL3`	*(version of 'foo.c' with version label 'REL3')*
`foo.c@@`	*(the element 'foo.c')*
`foo.c@@/main`	*(the main branch of element 'foo.c')*

For versions, `-version` overrides these interpretations of *pname*.

–ver·sion *version-selector*
(for use with versions only) For each *pname*, describes the version specified by *version-selector*. This option overrides both version-selection by the view and version-extended naming. See the *version_selector* manual page for syntax details.

–anc·estor (for use with elements and versions only) Describes the closest common ancestor version of all the *pname* arguments, which must all be versions of the same element. See the *merge* manual page for a discussion of "closest common ancestor".

–hli·nk *hlink-selector …*
One or more names in this form:

hyperlink-type-name@hyperlink-ID[*@pname-in-vob*]

Hyperlinks are not file system objects — you cannot specify them with shell wildcards. The final component is required only for a hyperlink in another VOB. Examples:

```
DesignFor@598f
RelatesTo@58843@/vobs/monet
```

–typ·e *type-name …*
Lists information about the type objects specified by the *type-name* argument(s). If there are multiple types with the same name (for example, a label type and a hyperlink type are both named *REL3*), all of them are listed.

–poo·l *pool-name …*
Lists information about the storage pools specified by the *pool-name* argument(s).

–vre·plica [**–vob** *pname-in-vob*] *replica-name …*
Lists information about a particular replica of the current VOB, or another VOB.

–vob *pname-in-vob*
With `–type`, `–pool`, `–hlink`, or `–vreplica`, this option specifies the VOB whose non-file-system object is to be described. The *pname-in-vob* can be any location within the VOB.

You can also use an option in this form to indicate a VOB object to be described.

EXAMPLES

- Describe the version of element *msg.c* selected by your view.

```
% cleartool describe msg.c
version "msg.c@@/main/1"
  created 08-Dec-93.12:12:55 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  element type: c_source
  Labels:
    REL6
    REL1
```

- Describe a branch of an element, specifying it with an extended pathname.

```
% cleartool describe util.c@@/main/rel2_bugfix
branch "util.c@@/main/rel2_bugfix"
  created 08-Dec-93.12:15:40 by Bev Jackson (test user) (jackson.dvt@oxygen)
  branch type: rel2_bugfix
  element type: text_file
```

```
  branched from version: /main/31
```

- Describe the label type *REL3*.

```
% cleartool describe –type REL3
label type "REL3"
  created 08-Dec-92.12:13:36 by Bev Jackson (test user) (jackson.dvt@oxygen)
  constraint: one version per branch
```

- Create a *Tested* attribute type and apply the attribute to the version of element *util.h* selected by your current view. Then, use *describe* to display the newly applied attribute value, and use the `–fmt` option to format the output.

```
% cleartool  mkattype –nc –default '"TRUE"' Tested
% cleartool  mkattr –default Tested util.h
% cleartool describe –aattr –all –fmt "Name: %Xn\nType of object: %m\n" util.h
Name: util.h@@/main/CHECKEDOUT
Type of object: version
  Attributes:
    Tested = "TRUE"
```

- Describe *ddft*, the current VOB's default derived object storage pool.

```
% cleartool describe –pool ddft
pool "ddft"
  created 15-Dec-93.09:34:00 by jenny.adm@oxygen
  "Predefined pool used to store derived objects."
  kind: derived pool
  pool storage global pathname "/net/oxygen/usr/vobstore/tut/tut.vbs/d/ddft"
  maximum size: 0  reclaim size: 0  age: 96
```

- Describe how the current view names an element that is named *hello.mod* in the *jackson_fix* view.

```
% cleartool describe –cview /view/jackson_fix/usr/hw/src/hello.mod
version "/usr/hw/src/hello.c@@/main/4"
  created 08-Dec-92.12:16:29 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  element type: text_file
```

- Describe the VOB containing the current working directory. List views with checkouts or derived objects in that VOB.

```
% cleartool describe –vob . –long
versioned object base "/usr/hw"
  created 15-Dec-93.09:34:00 by jenny.adm@oxygen
  "VOB dedicated to development of "hello, world" program"
  VOB storage remote host:path "oxygen:/usr/vobstore/tut/tut.vbs"
  VOB storage local pathname "/usr/vobstore/tut/tut.vbs"
  VOB ownership:
    owner jackson
    group dvt
  VOB holds objects from the following views:
    oxygen:/usr/vobstore/tut/old.vws [uuid 249356fe.d50f11cb.a3fd.00:01:56:01:0a:4f]
```

- Describe a hyperlink.

```
% cleartool describe –hlink Merge@516262@/vobs/proj
hyperlink "Merge@516262@/vobs/proj"
  created 14-Jul-93.16:43:35 by Bill Bo (bill.user@uranus)
  Merge@516262@/vobs/proj  /vobs/proj/lib/cvt/cvt_cmd.c@@/main/v1.1_port/8 ->
   /vobs/proj/lib/cvt/cvt_cmd.c@@/main/71
```

- Describe a derived object in the current view.

```
% cleartool describe –cview util.o
derived object "util.o@@11-Apr.12:03.33"
  created 11-Apr-94.12:03:33 by Anne Duvo (anne.dev@oxygen)
  references: 2 (shared)
  derived pool: ddft
  => saturn:/usr/anne/views/anne_main.vws
  => oxygen:/usr/jackson/views/jackson_proj2.vws
```

SEE ALSO

cleartool subcommands: chpool, lshistory, lspool, lstype, merge, mkpool, rmview
fmt_ccase, version_selector

NAME diff – compare versions of a text-file element or a directory

SYNOPSIS

diff [**–win·dow** | **–tin·y**] [**–dif·f_format** | **–ser·ial_format** | **–col·umns** *n*]
[**–opt·ions** *pass-through-opts*] [**–pre·decessor**] *pname* …

DESCRIPTION

Calls an element-type-specific program (the *compare method*) to compare the contents of two or more file elements, or two or more directory elements. Typically the files are versions of the same file element; a directory comparison *must* involve versions of the same directory element.

You can also use this command to compare ordinary text files.

Selection of a 'compare' Method

diff uses ClearCase's *type manager* mechanism to determine how to compare the specified objects. For example, if the objects to be compared are all versions of a *compressed_text_file* element, *diff* invokes the program named *compare* in the type-manager directory */usr/atria/lib/mgrs/z_text_file_delta*.

If none of the objects is a version of an element, *diff* uses the *text_file_delta* type manager, and displays a message:

```
cleartool: Warning: No type info, using text file type manager.
```

For most ClearCase predefined type managers, the *compare* program is a symbolic link to the *cleardiff* utility. For more information, see the *type_manager* manual page.

TEXT FILE COMPARISON ALGORITHM AND REPORT FORMAT

Text files are compared using a pairwise-differences algorithm, as illustrated in Figure 4.

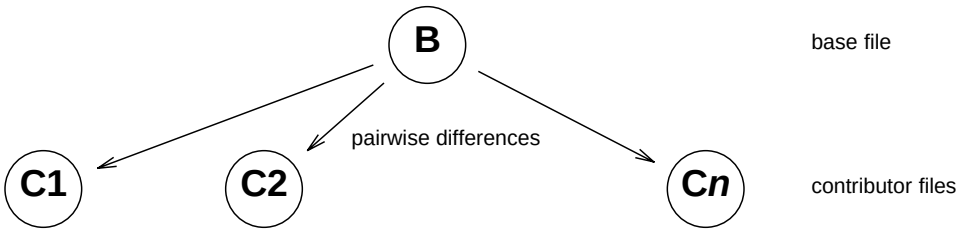


Figure 4. Pairwise-Differences Algorithm for Comparing Versions

- The first file is treated as the *base contributor file*. (It is referred to as `file 1` in command output, and simply as the *base file* in this manual page.)
- Each other *contributor file* is compared with the base file, producing a set of pairwise differences.

In effect, this process partitions the base file into two kinds of sections (groups of text lines): the unchanged sections, in which none of the other contributors differs from the base file; and the *difference sections*, in which one or more of the other contributors differs from the base file.

Each difference section is reported as one or more pairwise differences. For example, if three contributor files all differ from the base file in a particular section, *diff* lists the file1-file2 difference, followed by the file1-file3 difference, followed by the file1-file4 difference.

Side-by-Side File Comparison Report Style

The default file-comparison report begins with a *file summary*, which lists all the input files and their assignments as `file 1`, `file 2`, and so on. If no differences are detected among the files, this listing is replaced by the message `Files are identical`.

The remainder of the report is a series of pairwise *differences*, each of which is preceded by a descriptive *header* line, as illustrated in Figure 5.

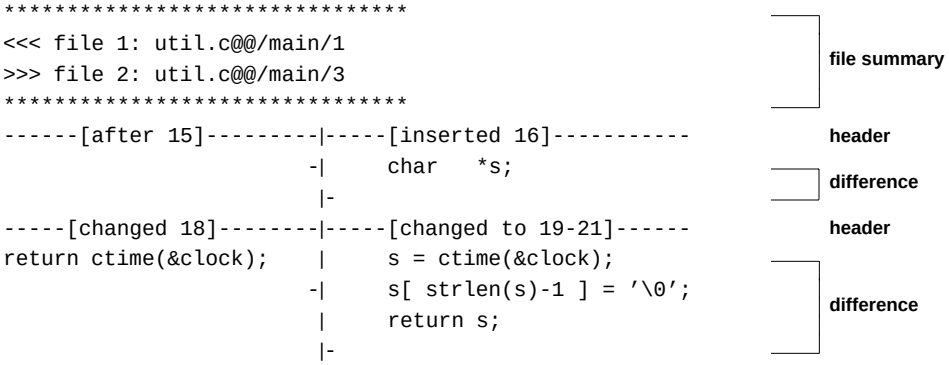


Figure 5. Side-by-Side File-Comparison Report

The `-quiet` and `-diff_format` options suppress the file summary. The `-headers_only` option suppresses the differences, listing the header lines only.

Header Lines. Each header line indicates which text lines in the input files were changed, and how they were changed. The words describe the change in terms of "how the first file was changed to produce the second file". The numbers in the header lines are the same (and in the same order) as in the headers output by standard UNIX *diff*. Header lines can have the following formats, where each of *A*, *B*, and so on might be a single line number (for example, `46`) or a range (for example, `256-290`):

`------------[after` *A*`]------------|------------[inserted` *B*`]-------------`

Insertion of one or more lines. *B* indicates where the inserted lines occur in the second file. *A* indicates the corresponding point in the first file.

`-----------[deleted` *C*`]-----------|-------------[after` *D*`]--------------`

Deletion of one or more lines. *C* indicates which lines from the first file were deleted. *D* indicates the corresponding point in the second file.

`--------[deleted/moved C]--------|----------[after D now B]-----------`

Deletion of one or more lines from the first file, to which there corresponds an insertion of the same lines in the second file. Typically, this indicates that a range of lines was moved from one location to another — see `inserted/moved` below. *C* indicates where the lines were deleted from the first file; *B* indicates the location where these same lines were inserted in the second file. *D* indicates the point in the second file that corresponds to *C*.

`-----------[changed X]-----------|------------[changed to Y]-----------`

One or more lines changed in place. *X* indicates which lines in the first file were changed. *Y* indicates where the replacement lines occur in the second file.

`----------[after A was C]--------|----------[inserted/moved B]--------`

Insertion of one or more lines in the second file, to which there corresponds a deletion of the same lines from the second file. Typically, this indicates that a range of lines was moved from one location to another — see `deleted/moved` above. *B* indicates where the lines were inserted in the second file; *C* indicates where these same lines were deleted from the first file; *A* indicates the point in the first file that corresponds to *B*.

Differences. *diff* can report a pairwise difference in several ways. When comparing files, its default is to list corresponding lines side-by-side, and possibly truncated.

A plus sign (+) at the end of a difference line indicates that it has been truncated in the report. To see more of such lines, you can increase the report width using the `–columns` or `–tiny` option. The minus signs (-) along the vertical separator line indicate the endpoints of the groups of differing lines. They help to distinguish empty lines in the input files from blank space in command output.

Other File Comparison Report Styles

The `–serial_format` option causes the differences to be reported as entire lines, in above-and-below format instead of side-by-side format. Compare the following with the example above:

```
-----[after 15 inserted 16]-----
>     char   *s;
-----[18 changed to 19-21]-----
<     return ctime(&clock);
---
>     s = ctime(&clock);
>     s[ strlen(s)-1 ] = ' ';
>     return s;
```

The `–diff_format` option causes both the headers and differences to be reported in the style of the standard UNIX *diff(1)* utility. Compare the following with the examples above:

```
15a16
>     char   *s;
18c19,21
<     return ctime(&clock);
---
>     s = ctime(&clock);
>     s[ strlen(s)-1 ] = ' ';
>     return s;
```

When *diff* compares multiple files, it adds file-identification annotations to the *diff*-style headers.

DIRECTORY-COMPARISON ALGORITHM AND REPORT FORMAT

For a comparison of directory versions, the *cleardiff* program is not invoked at all. Instead, *diff* invokes a directory-element-specific *compare* method, whose report format is very similar to *cleardiff*'s, described above.

(This program, */usr/atria/lib/mgrs/directory/compare*, can be invoked only by *cleartool* subcommands.)

Kinds of Directory Entries

A version of a ClearCase directory can contain several kinds of entries:

- **File Elements** — reported by *diff* as: (1) the element's name (in this directory version); (2) the element's creation time; (3) the username of the element's creator. Example:

  ```
  obj2  12-Aug.14:00 akp
  ```

  Note that multiple *VOB hard links* to the same element will have the same creator and creation time, but different names.

- **Directory Elements** — reported by *diff* in the same way as file elements, except that a slash character (`/`) is appended to the element name. Example:

  ```
  sub6/  13-Aug.15:00 akp
  ```

- **VOB Symbolic Links** — reported by *diff* as: (1) the link's name (in this directory version), followed by `->` and the text (contents) of the link; (2) the link's creation time; (3) the username of the element's creator. Example:

  ```
  doctn -> ../vobs/doctn 13-Aug.08:44 akp
  ```

How Differences are Reported

diff's report is a series of differences, each of which focuses on one directory entry. A difference can be a simple addition or deletion; it can also involve the renaming of an existing object, or the reuse of an existing name for another object. The following examples illustrate the various possibilities:

```
----------------------------------------|---------------[ added ]----------
                                       -| obj2  12-Aug.14:00 akp
```

An object named *obj2* was added (*mkelem*, *mkdir*, or *ln*) in the second version of the directory.

```
---------------[ removed ]--------------|---------------------------------
obj5  12-Aug.14:00 akp                  |-
```

An object named *obj5* was removed (*rmname*) in the second version of the directory.

```
---------------[ renamed ]--------------|-------------[ renamed to ]-------
obj3  12-Aug.14:00 akp                  | obj3.new  12-Aug.14:00 akp
```

An object named *obj3* was renamed (*mv*) to *obj3.new* in the second version of the directory.

```
-------------[ old object ]-------------|-------------[ new object ]-------
obj4  12-Aug.14:04 akp                  | obj4  19-Oct.17:10 akp
```

In the second version of the directory, an object named `obj4` was removed (*rmname*) and another object was created with that same name.

```
----------[ old link text ]------------|-----------[ new link text ]------
doctn -> ../vobs/doctn 13-Aug.08:44 akp | doctn -> ../vb/doctn 19-Sep.21:01 akp
```

(special case of the preceding example) In the second version of the directory, a VOB symbolic link named `doctn` was removed (*rmname*) and another VOB symbolic link was created with that same name.

```
---------------[ renamed ]-------------|--------[ renamed to ]------------
obj4  12-Aug.14:01 akp                 | obj1  12-Aug.14:01 akp
---------------[ removed ]-------------|---------------------------------
obj1  12-Aug.14:00 akp                 |-
```

These two differences show that in the second version of the directory, an object named `obj1` was removed and another object was renamed from `obj4` to `obj1`.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

With the exception of `–predecessor` and `–options`, *diff* options are the same as those of *cleardiff*.

Using a Separate Window. *Default:* Sends output to the current window.

–win·dow Displays output in a separate difference window, formatted as with `-columns 120`. Type an operating system `INTR` character (typically, `<Ctrl-C>`) in the difference window to close it. The *diff* command returns immediately, not waiting for the difference window to be closed.

–tin·y Same as `–window`, but uses a smaller font in a 165-character difference window.

Output Format. *Default:* Reports differences in the format described in "How Differences Are Reported" above.

–ser·ial_format
Reports differences with each line containing output from a single file, instead of in a side-by-side format.

–col·umns *n*
Establishes the overall width of a side-by-side report. The default width is 80 — only the first 40 or so characters of corresponding difference lines appear. If *n* does not exceed the default width, this option is ignored.

–dif·f_format
Reports both headers and differences in the same style as the standard *diff*(1) utility, and suppresses the file summary from the beginning of the report.

Passing Through Options to the 'compare' Method. *Default:* Does not pass any special options to the underlying *compare* method (typically, the *cleardiff* program).

–opt·ions *pass-through-opts*
Specifies one or more *compare* method options that are not directly supported on the *diff* command line. Use quotes if you are specifying more than one pass-through option — *diff* must see them as a single command-line argument. For example, this command passes through the `–quiet` and `–blank_ignore` options:

```
cleartool diff -options "-qui -b" -pred util.c
```

Comparison of a Version with its Predecessor. *Default:* None.

–pre·decessor
Effectively converts the first *pname* argument into two names: (1) the *predecessor* of *pname* in the version tree; (2) *pname* itself. If *pname* specifies a checked-out version, the predecessor is the version from which it was checked out.

An error occurs if the *pname* does not specify a version:

```
cleartool: Error: Not a vob object: "myfile.c".
```

Specifying the Data to be Compared. *Default:* None.

pname … One or more pathnames, indicating the objects to be compared: versions of file elements, versions of directory elements, or any other files. If you don't use `–predecessor`, you must specify at least two *pname* arguments.

EXAMPLES

- Compare the version of a file element in the current view with the version in another view.
  ```
  % cleartool diff util.c /view/jackson_old/usr/hw/src/util.c
  ```
- Compare the version of *foo.c* in the current view with its predecessor version:
  ```
  % cleartool diff –predecessor foo.c
  ```
- Compare three files: the version of *msg.c* selected by the current view, its predecessor version, and *msg.SAVE* in your home directory:
  ```
  % cleartool diff –pre msg.c $HOME/msg.SAVE
  ```
- In a separate 132-column window, compare the version of *util.c* in the current view with a version on the *rel2_bugfix* branch.
  ```
  % cleartool diff –window –columns 132 util.c util.c@@/main/rel2_bugfix/LATEST
  ```

SEE ALSO

cleartool subcommands: diffcr, merge, xdiff, xmerge
cleardiff, xclearcase, xcleardiff, type_manager

NAME diffcr – compare configuration records created by clearmake or clearaudit

SYNOPSIS

diffcr [**–r·ecurse** | **–fla·t**] [**–sel·ect** *do-leaf-pattern*] [**–ci**] [**–ele·ment_only**]
[**–vie·w_only**] [**–typ·e** { **f** | **d** | **l** } …] [**–nam·e** *tail-pattern*]
[**–wd**] [**–nxn·ame**] [**–l·ong** | **–s·hort**] *do-pname-1 do-pname-2*

DESCRIPTION

Compares the *configuration records* (CRs) of two *derived objects*. A CR is produced by *clearmake* when it finishes executing a build script. By comparing CRs, you can determine differences in:

- versions of *MVFS objects* used as sources or produced during the build (includes elements and other objects whose pathnames are under a VOB mount point)
- versions of non-MVFS objects that appeared as *makefile dependencies* during the build (explicit dependencies declared in the makefile)
- the total number of times an object was referenced during a build, and the first target in which that object was referenced
- build options (which can come from the command line, the UNIX environment, the makefile itself, and so on)
- the build script executed
- non-critical differences, such as the date/time of the build, view name, host name, and so forth

The *do-pname* arguments specify the derived objects to be compared. You can use a derived object ID (DO-ID) to identify a derived object created in any view. Alternatively, you can use a standard or view-extended pathname (for example, *myprog.o* or */view/jpb/usr/src/myprog.o*), to identify a DO created in the current view or another view. A DO-ID takes the form:

*DO-pname***@@***creation_date***.***creation_time***.***id-number*

Example: `myprog.o@@11-Nov.17:39.3871`. (The *lsdo* command lists derived objects by DO-ID.)

diffcr supports the same filter and report style options as the *catcr* command. This means that you can restrict the comparison to particular subtargets of the *do-pnames*, control which objects actually appear in the listing, select how pathnames are displayed, and expand the listing to include comments and other supplementary information. See *catcr* for additional background.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Comparing Differences in Subtargets. *Default: diffcr* compares the CRs for *do-pname-1* and *do-pname-2* only, not for any of their subtargets.

–r·ecurse Compares the CRs of the two specified derived objects, and their common subtargets. Each pair of CRs is compared separately. By default, a recursive comparison does not descend into DO versions; use `–ci` to override this.

–fla·t Similar to `–recurse`, but consolidates the CRs for each *do-pname-n* into a single list, with no duplicates, and then compares the lists. The report includes file system objects only; no headers, variables and options, or build scripts. It also includes the total number of times each object was referenced during the build, and the first target in which that object was referenced (`First seen in target`).

–sel·ect *do-leaf-pattern*
Starts the comparison at the subtarget(s) of *do-pname* that match *do-leaf-pattern* (which can include pattern-matching characters — see the *wildcards_ccase* manual page). This option is useful for focusing on a particular object (for example, object module `hello.o`) that was built as part of a larger object (for example, executable `hello`).

–ci (for use with
`–recurse` or `–flat` only) Descends into the CRs of DO versions that were used as build sources.

Specifying Kinds of Objects to Display. *Default: diffcr* reports on all objects in the CRs, which may include: source files, directories, and symbolic links; derived objects; makefiles; view-private files, and non-MVFS objects that were explicitly declared as dependencies.

–ele·ment_only
Lists versions of elements only, including checked-out versions. This option excludes from the listing derived objects (except DO versions), view-private files and directories, symbolic links, and non-MVFS objects.

–vie·w_only
Lists view-private objects only, including checked-out versions of elements. If you specify this option along with `–element_only`, the listing includes just checked-out versions of elements.

–typ·e { **f** | **d** | **l** } …
Lists file system objects of a particular kind: files (`f`) directories (`d`), or links (`l`). The default value varies with the report style: normal and short listings (`–short`) default to `f`; long listings (`–long`) default to `fdl`. You may specify multiple kinds of objects by grouping them into a single argument; `-type fd`, for example.

–nam·e *tail-pattern*
Considers the entry for a file system object only if its final pathname component matches the specified pattern. See the *wildcards_ccase* manual page for a list of pattern-matching characters.

Controlling Report Appearance. *Default: diffcr* reports, in three sections, on MVFS objects, variables and options, and the build script. The report uses full pathnames, and it omits comments and directory versions.

–l·ong Expands the report to include the kinds of objects in the CR, and comments. With –`makefile`, adds comments only. For example, an object may be listed as a *version*, a *directory version*, or *derived object* (see `ls –long` for a complete list). Comments indicate if an object is `in makefile`, a `referenced derived object`, or a `new derived object`.

–s·hort Restricts the report to file system objects only (omits header information, variables and options, and build scripts). With –`makefile`, the listing also includes build scripts.

–wd Lists pathnames relative to the current working directory, rather than as full pathnames.

–nxn·ame Lists simple pathnames for MVFS objects, rather than version-extended pathnames or DO-IDs.

Specifying the Derived Objects. *Default:* None.

do-pname-1, *do-pname-2*
Standard pathnames and/or DO-IDs of two derived objects to be compared. Either or both can be a DO version.

EXAMPLES

- Compare the CRs of two derived objects built at the name *bgrs*. Use *lsdo* to determine the DO-ID of the derived object that is not visible in the current view.

```
% cleartool lsdo –zero bgrs
11-Dec.15:24   "bgrs@@11-Dec.15:24.1487"
11-Dec.12:05   "bgrs@@11-Dec.12:05.1256"
```

```
% cleartool diffcr bgrs bgrs@@11–Dec.12:05.1956
< Target bgrs built by jones.dvt
> Target bgrs built by jones.dvt
< Reference Time 11-Dec-93.15:23:52, this audit started 11-Dec-93.15:23:59
> Reference Time 11-Dec-93.12:02:39, this audit started 11-Dec-93.12:04:52
< View was oxygen:/usr/jones/views/main.vws [uuid 66e68edc.471511cd.ac55.08:00:2b:33:ec:ab]
> View was oxygen:/usr/jones/views/r1_fix.vws [uuid 8b468fd0.471511cd.aca5.08:00:2b:33:ec:ab]
----------------------
----------------------------
MVFS objects:
----------------------------
< /vobs/docaux/bgr/sun4/bgrs@@11-Dec.15:24.1987
> /vobs/docaux/bgr/sun4/bgrs@@11-Dec.12:05.1956
----------------------------
< /vobs/docaux/bgr/sun4/bugs.o@@11-Dec.15:23.1981
> /vobs/docaux/bgr/sun4/bugs.o@@11-Dec.12:03.1902
----------------------------
< /vobs/docaux/bgr/sun4/bugsched.o@@11-Dec.15:23.1984
> /vobs/docaux/bgr/sun4/bugsched.o@@11-Dec.12:04.1953
```

The comparison shows that the builds used different versions of the object modules *bugs.o* and *bugsched.o*.

- Compare the same two derived objects again, this time including the CRs of all subtargets.

```
% cleartool diffcr –flat bgrs bgrs@@11–Dec.12:05.1956
----------------------------
MVFS objects:
----------------------------
----------------------------
< First seen in target "bugs.o"
```

```
<     1 /vobs/docaux/bgr/bugs.c@@/main/2        <11-Dec-92.15:22:53>
> First seen in target "bugs.o"
>     1 /vobs/docaux/bgr/bugs.c@@/main/1        <19-Dec-91.11:49:54>
---------------------------
< First seen in target "bugsched.o"
<     1 /vobs/docaux/bgr/bugsched.c@@/main/2    <11-Dec-92.15:23:04>
> First seen in target "bugsched.o"
>     1 /vobs/docaux/bgr/bugsched.c@@/main/1    <19-Dec-91.11:50:07>
---------------------------
< First seen in target "bgrs"
<     1 /vobs/docaux/bgr/sun4/bgrs@@11-Dec.15:24.1987
> First seen in target "bgrs"
>     1 /vobs/docaux/bgr/sun4/bgrs@@11-Dec.12:05.1956
---------------------------
< First seen in target "bgrs"
<     2 /vobs/docaux/bgr/sun4/bugs.o@@11-Dec.15:23.1981
> First seen in target "bgrs"
>     2 /vobs/docaux/bgr/sun4/bugs.o@@11-Dec.12:03.1902
---------------------------
< First seen in target "bgrs"
<     2 /vobs/docaux/bgr/sun4/bugsched.o@@11-Dec.15:23.1984
> First seen in target "bgrs"
>     2 /vobs/docaux/bgr/sun4/bugsched.o@@11-Dec.12:04.1953
```

The integer at the beginning of an entry indicates the number of times the object was referenced during the build. The `first seen in target` message indicates the first target rebuild in which the object was referenced.

- For the same two derived objects as in the preceding examples, compare the file element versions used to build subtarget *bugsched.o*. Report the differences in short format.

```
% cleartool diffcr –short –select bugsched.o –type f –element_only \
     bgrs bgrs@@11–Dec.12:05.1956
---------------------------
< /vobs/docaux/bgr/bugsched.c@@/main/2
> /vobs/docaux/bgr/bugsched.c@@/main/1
```

- Compare two builds of program *main*, listing only those entries that involve files *src/prog.c*, *include/prog.h*, and *bin/prog.o*.

```
% diffcr –recurse –name 'prog.[cho]' main1 main2
```

SEE ALSO

cleartool subcommands: ls, lsdo, catcr, rmdo
clearaudit, clearmake, wildcards_ccase

NAME edcs – edit config spec of a view

SYNOPSIS

edcs [**–tag** *view-tag*] [*file*]

DESCRIPTION

Revises a view's config spec by invoking a text editor on an existing config spec:

- the view's current config spec, or
- a text file that you would first like to edit, then make the view's config spec (if you don't need to edit the file, just use *setcs*)

At the end of the edit session, there is a confirmation step:

```
Compile and set this config spec? [yes]
```

A `no` answer cancels the command — the view retains its current config spec.

The text editor invoked by *edcs* is specified by the environment variable VISUAL (first choice) or EDITOR (second choice). If neither of these EVs is set, *vi*(1) is invoked.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the View. *Default:* Edits and sets a config spec for the current view.

–tag *view-tag*
The view-tag of any view.

Specifying the Config Spec File. *Default:* Edits the view's current config spec, which is stored in file *config_spec* in the view storage directory.

file The pathname of a file to be used as input to the edit session. If the file does not exist, *edcs* creates it.

EXAMPLES

- Edit the config spec of the current view.
  ```
  % cleartool edcs
  ```
- Edit the config spec of the view with the view-tag *jackson_fix*.
  ```
  % cleartool edcs –tag jackson_fix
  ```
- Use an ASCII file named *cspec_REL3* as input to an edit session, producing a new config spec for the current view.
  ```
  % cleartool edcs cspec_REL3
  ```

SEE ALSO

cleartool subcommands: catcs, lsview, mktag, setcs
config_spec

NAME find – use pattern, query, or expression to search for objects

SYNOPSIS

- Find objects visible in currently-selected directory structure:

  find *pname …* **[–nam·e** *pattern* **] [–fol·low] [–dep·th | –nr·ecurse | –d·irectory]**
  selection-options action-options

- Find all objects in named VOBs:

  find *pname-in-vob …* **–a·ll [–vis·ible | –nvi·sible]** *selection-options action-options*

- Find objects throughout all mounted VOBs:

  find –avo·bs [–vis·ible | –nvi·sible] *selection-options action-options*

  selection-options:
  –use·r *login-name*
  –gro·up *group-name*
  –typ·e { **f** | **d** | **l** } …
  –nxn·ame
  –cvi·ew
  –ele·ment *query*
  –bra·nch *query*
  –ver·sion *query*

  action-options (at least one required, multiple allowed):
  –pri·nt
  –exe·c *command-invocation*
  –ok *command-invocation …*

DESCRIPTION

The ClearCase *find* command is similar to the standard UNIX *find*(1) command. Only a limited set of the standard *find* options are supported; the way that commands are invoked on selected objects (–exec and –ok options) differs from the UNIX standard.

find starts with a certain set of objects, selects a subset of the objects, and then performs an action on the subset. The selected objects can be elements, branches, versions, or VOB symbolic links. The action can be to list the objects, or to execute a command on each object, either conditionally or unconditionally.

Typically, you start with all objects in a directory tree as seen through your view (similar to standard UNIX *find*). You can also start with *all* objects in one or more VOBs, regardless of their visibility through a particular view.

OPTIONS AND ARGUMENTS

Specifying the Starting Set of Objects. *Default:* None — you must specify one of the following: (1) one or more elements, using *pname* arguments; (2) one or more VOBs, using *pname* arguments along with the `–all` option; (3) all mounted VOBs, using the `–avobs` option.

pname … One or more file and/or directory elements; *find* starts with the elements, branches, and versions that are part of the specified file elements and the subtrees under the specified directory elements.

–a·ll Modifies the meaning of each *pname* argument to specify its entire VOB, not just a single file or directory.

–avo·bs *find* starts with all the elements, branches, and versions in all the VOBs active (mounted) on the local host. (If environment variable CLEARCASE_AVOBS is set to a colon-separated list of VOB-tags, this set of VOBs is used instead.)

NOTE: Processing all of a VOB's elements (using `–all` or `–avobs`) is an order of magnitude faster than "walking" its entire directory tree (by specifying the VOB's root directory as a *pname* argument). With these options, the order in which elements are processed and/or reported is very different from "tree-walk" order.

Considering Objects that are Not Currently Visible. *Default:* If you specify one or more entire VOBs, using `–all` or `–avobs`, all elements in the VOB(s) are included, whether or not they are visible in the view.

–vis·ible Includes only those elements, along with their branches and versions, that are visible (have a standard pathname) in the view.

–nvi·sible Includes only those elements, along with their branches and versions, that are not visible (do not have a standard pathname) in the view.

Selecting Elements Using Standard Criteria. The following options use standard-UNIX criteria to select subsets of objects. See also the following section, where the options involve ClearCase-specific criteria.

–nam·e *pattern*
Selects the subset of objects whose element names match the specified file name pattern. (See the *wildcards_ccase* manual page.)

–fol·low Causes VOB symbolic links to be traversed during the "walk" of the directory tree.

–dep·th (same as standard UNIX) Causes directory entries to be processed before the directory itself.

–nr·ecurse For each directory element, selects the objects in the element itself, and in the file and directory elements within it, but does not descend into its subdirectories.

–d·irectory For each directory, examines only the directory itself, not the file elements it catalogues.

–use·r *login-name*
Selects only those objects in the subset of elements owned by user *login-name*.

–gro·up *group-name*
Selects only those objects in the subset of elements belonging to group *group-name*.

–typ·e f
–typ·e d
–typ·e l Selects the subset of objects of a certain kind: file elements (**f**), directory elements (**d**), or VOB symbolic links (**l**). To include multiple kinds of objects, group the keyletters into a single argument (`-type fd`), or use multiple options (`-type f -type d`).

Selecting Elements Using Queries. The options in this section select a subset of objects using the ClearCase *query language*, which is described in the *query_language* manual page. You can use these options in any combination. They are always applied in this order, successively refining the set of selected objects: first `–element`, then `–branch`, then `–version`. The result of applying one or more of these options is a set of objects at the "finest granularity level" — all versions if you used `–version`, or else all branches if you used `–branch`, or else all elements if you used `–element`. If you use none of these options, the set includes elements and VOB symbolic links. There is no way to use a query to select a set of VOB symbolic links.

The "Formulating Queries" section below provides additional usage notes.

–ele·ment *query*
Selects element objects using a ClearCase query; all of a selected element's branches and versions are also selected.

–bra·nch *query*
From the set of objects that "survived" the element-level query (if any), selects branch objects using a ClearCase query; all of a selected branch's versions are also selected.

–ver·sion *query*
From the set of objects that "survived" the element-level and branch-level queries (if any), selects version objects using a ClearCase query.

–cvi·ew Modifies the set of objects selected by the `–element`, `–branch`, and `–version` queries (if any):

- If you did not specify `–version`, replaces each element and branch with the version that is currently selected by the view. (No substitution is performed on VOB symbolic links.)
- If you did specify `–version`, further restricts the subset to versions that are currently selected by the view.

Use of Extended Pathnames. *Default: find* submits the objects it selects to the specified action using extended pathnames, such as *foo.c@@* (element), *foo.c@@/main* (branch), or *foo.c@@/main/5* (version).

–nxn·ame Removes the extended naming symbol (by default, `@@`) and any subsequent version-ID or branch pathname from the name of each selected object. Duplicate names that result from this transformation are suppressed. In effect, this option transforms ClearCase extended names into standard UNIX names; it also transforms names of branches or versions into names of elements.

Specifying the Action. *Default:* None — you must specify an action to be performed on the selected objects. You can specify a sequence of several actions, using two `-exec` options, or `-exec` followed by `-print`, and so on.

-pri·nt Lists the names of the selected objects, one per line.

-exe·c *command-invocation*
Execute the specified command once for each selected object.

-ok *command-invocation*
For each selected object, display a confirmation prompt; if you respond "yes", execute the specified command.

The `-exec` and `-ok` command invocation does not follow the standard UNIX *find* syntax. Do not use `{}` to indicate a selected object, and a quoted or escaped semicolon to terminate the command. Instead, enter the entire command as a quoted string; use one or more of these environment variables to reference the selected object:

CLEARCASE_PN
pathname of selected element or VOB symbolic link

CLEARCASE_XN_SFX
extended naming symbol (default: `@@`)

CLEARCASE_ID_STR
branch pathname of a branch object (*/main/rel2_bugfix*); version-ID of a version object (*/main/rel2_bugfix/4*); null for a version object

CLEARCASE_XPN
full version-extended pathname of the selected branch or version (concatenation of the three preceding variables)

FORMULATING QUERIES

The meaning of a query predicate often depends on which option you use it with. For example, an `attype` query will be TRUE with `-element` if the element itself has an attribute of type *attype-name*; with `-branch`, it will be TRUE if the branch itself has an attribute of type *attype-name*; and with `-version`, it will be TRUE if the version itself has an attribute of type *attype-name*. Other query predicates have similar object-dependent interpretations.

In general, you should quote each query to prevent shell-level interpretation of characters such as `(`, `"`, and `<Space>`.

In *csh*(1), it is not sufficient to quote the ! character; you have to escape it:

```
% cleartool find . -version '\! attype(TESTED)' -print
```

Unlike queries in version selectors, *find* queries do not need to be enclosed in braces (`{ … }`).

See the *query_language* manual page for complete details on query language primitives and syntax.

EXAMPLES

- List all file elements in and below the current working directory.

```
% cleartool find . –type f –print
./Makefile@@
./hello.c@@
./hello.h@@
./msg.c@@
./util.c@@
```

  This listing includes the extended naming symbol. The `–nxname` option suppresses this symbol.

- List the version labeled *REL1* for each element in or below the current working directory.

```
% cleartool find . –version "lbtype(REL1)" –print
.@@/main/1
./Makefile@@/main/1
./hello.c@@/main/2
```

- List each header file (`*.h`) and C source file (`*.c`) for which some version is labeled *REL2* or *REL3*.

```
% cleartool find . –name '*.[ch]' \
      –element 'lbtype_sub(REL2) || lbtype_sub(REL3)' –print
./hello.c@@
./hello.h@@
./util.c@@
```

- List all versions that have a *QAed* attribute with the string value `"Yes"`. Note the use of double-quotes inside single-quotes to specify the string literal.

```
% cleartool find . –version 'QAed == "YES"' –print
./Makefile@@/main/2
./hello.c@@/main/4
./hello.h@@/main/1
./util.c@@/main/2
./util.c@@/main/rel2_bugfix/1
```

- List the standard name of each element that has (or contains a branch or version that has) a *BugNum* attribute with the value *189*.

```
% cleartool find . –nxname –element 'attr_sub(BugNum,==,189)' –print
./hello.c
```

- For each element that has had a merge from the *rel2_bugfix* branch to the *main* branch, archive the current version of the element to a *tar*(1) file in your home directory.

```
% cleartool find . –element "merge(/main/rel2_bugfix,/main)" \
      –exec 'tar –cvf $HOME/rel2bugmerge.tar $CLEARCASE_PN'
```

- If any element's most recent version on the main branch is missing label *REL3*, label it.

```
% cleartool find . –version 'version(/main/LATEST) && \! lbtype(REL3)' \
      –exec 'cleartool mklabel –replace REL3 $CLEARCASE_XPN'
```

- Attach a *Testing* attribute with string value `"Done"` to all versions labeled *REL2*. Note that the double-quote characters that surround the string value must themselves be escaped or quoted:

```
% cleartool find . –ver 'lbtype(REL2)' \
      –exec 'cleartool mkattr Testing \"Done\" $CLEARCASE_XPN'
```

- Conditionally delete all branches of type *experiment*.

```
% cleartool find . –branch 'brtype(experiment)' \
      –ok 'cleartool rmbranch –force $CLEARCASE_XPN'
```

- Change all elements currently using storage pool *my_cpool* to use pool *cdft* instead.

```
% cleartool find . –all –element 'pool(my_cpool)' \
      –exec 'cleartool chpool cdft $CLEARCASE_PN'
```

- Obsolete elements that are no longer visible.

```
% cleartool find . –all –nvisible \
      –exec 'cleartool lock –obsolete $CLEARCASE_PN'
```

SEE ALSO

cleartool subcommands: describe, ls
find(1), query_language, wildcards_ccase

NAME findmerge – search for elements that require a merge / optionally perform merge

SYNOPSIS

findm·erge { **–avo·bs** | *pname …* | [*pname …*] **–all** }
[**–fol·low**] [**–dep·th** | **–nr·ecurse** | **–d·irectory**] [**–nam·e** *pattern*]
[**–typ·e** { **f** | **d** | **fd** }] [**–ele·ment** *query*]
[**–use·r** *login-name*] [**–gro·up** *group-name*]
{ **–fta·g** *view-tag* | **–fve·rsion** *version-selector* | **–fla·test** }
[**–nze·ro**] [**–nba·ck**] [**–why·not**] [**–vis·ible**] [**–log** *pname*]
{
–pri·nt [**–l·ong** | **–s·hort** | **–nxn·ame**]
| **–exe·c** *command-invocation* | **–ok** *command-invocation*
| **–mer·ge** | **–okm·erge** | **–xme·rge** | **–okx·merge**
} …
[**–abo·rt** | **–qal·l**] [**–ser·ial**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]

DESCRIPTION

For one or more elements, determines whether a merge is required *from* a specified version *to* the version selected by your view, then executes one or more actions:

- listing the elements that require a merge
- performing the required mergers (checking out elements as necessary)
- performing an arbitrary command

findmerge works as follows:

1. It considers a set of elements, which you specify using syntax similar to that of the standard UNIX *find*(1) command and the ClearCase *find* command.
2. For each of these elements, *findmerge* examines the relationship between the version selected by your view and the version specified by the `–ftag`, `–fversion`, or `–flatest` option. It determines whether or not a merge is required *from* that other version *to* your view's version.
3. *findmerge* then performs the action(s) you specify with `–print`, `–exec`, and/or the various `–merge` variants.

PERMISSIONS AND LOCKS

Any user can enter a *findmerge* command. If the specified action involves a *checkout* and/or *merge*, then the permissions-checking of those commands comes into play.

OPTIONS AND ARGUMENTS

Specifying the Elements To Be Considered. *Default:* None.

–avo·bs Considers all the elements in all the VOBs active (mounted) on the local host. (If environment variable CLEARCASE_AVOBS is set to a colon-separated list of VOB-tags, this set of VOBs is used instead.)

pname … One or more file and/or directory elements; only the specified file elements and the subtrees under the specified directory elements will be considered.

pname … **–all**
Appending `–all` to a *pname* list causes all the elements in the VOB(s) containing the *pname*(s) to be considered, whether or not they are visible in your view.

Narrowing the List of Elements To Be Considered. Use the following options to select a subset of the elements specified by *pname* arguments and the `–all` or `–avobs` option.

–dep·th (same as standard UNIX) Causes directory entries to be processed before the directory itself.

–fol·low Causes VOB symbolic links to be traversed.

–nr·ecurse For each directory element, considers the file and directory elements within it, but does not descend into its subdirectories.

–d·irectory For each directory, considers only the directory itself, not the file elements it catalogues.

–nam·e *pattern*
Considers only those elements whose leaf names match the specified file name pattern. (See the *wildcards_ccase* manual page.)

–typ·e f
–typ·e d
–typ·e fd Considers file elements only (**f**), directory elements only (**d**), or both (**fd**).

–ele·ment *query*
Considers only those elements that satisfy the specified query (same as the ClearCase *find* command).

–use·r *login-name*
Considers only those elements owned by user *login-name*.

–gro·up *group-name*
Considers only those elements belonging to group *group-name*.

Specifying the 'FROM' Version. *Default:* None — you must use one of these options to specify another version of each element, to be compared with the version selected by your view.

–fta·g *view-tag*
Compare with the version selected by the view with the specified view-tag. A version of the same element is always used, even if the element has a different name in the other view.

–fve·rsion *version-selector*
Compare with the version specified by the version-selector.

–fla·test (consider only elements that are currently checked-out) Compare with the most recent version on the branch from which your version was checked out. This option is useful with elements for which you have *unreserved* checkouts: if one or more new versions have been checked in by other users, you must merge the most recent one into your checked-out version before you can perform a checkin.

Special Version Tree Geometry: Merging From Version 0. If a merge is required from a version that happens to be version 0 on its branch, *findmerge*'s default behavior is to perform the merge and issue a warning message:

```
Element "util.c" has empty branch [to /main/6 from /main/br1/0]
```

More often, *findmerge* determines that no merge is required from a zeroth version; it handles this case just like any other no-merge-required case. Figure 6 illustrates both these cases.

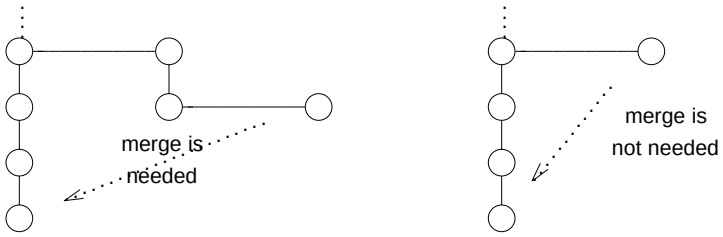


Figure 6. Merging From the Zeroth Version on a Branch

The following option overrides this default behavior.

–nze·ro Does not perform a merge if the "from" contributor is version 0 on its branch. This gives you the opportunity to delete the empty branch, and then perform a merge from the version at which the branch was created.

Special Version Tree Geometry: Merge Back-and-Out to Subbranch. *findmerge* flags this special case with a warning message:

```
Element "msg.c" requests merge to /main/12 backwards on same branch from /main/18
```

This situation arises when:

- You are merging from a "parent" branch to a subbranch.
- For a particular element, no subbranch has been created yet.
- Your view selects a version of that element using a `-mkbranch` config spec rule.

In this case, *findmerge*'s default behavior is to perform the merge by checking out the element (which creates the subbranch at the "to" version), then overwriting the checked-out version with the "from" version. Figure 7 illustrates this case.

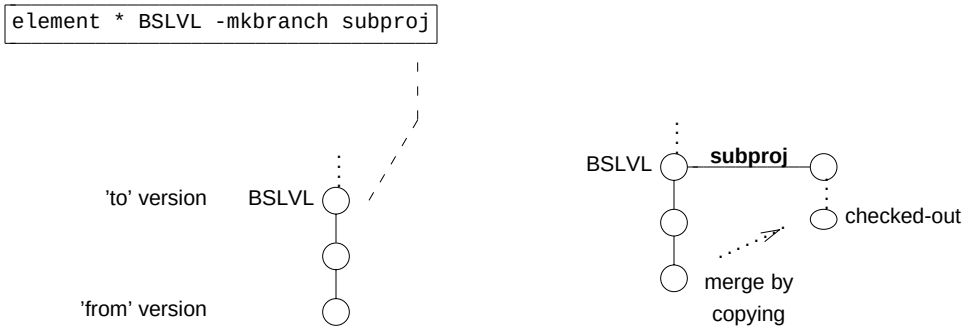


Figure 7. Merging Back and Out to a Subbranch

The following option overrides this default behavior.

–nba·ck Does not perform the merge in the case described above. It may be appropriate to simulate the merge by moving the version label down to the "from" version. Note, however, that this alternative leaves the element without a subbranch, which may or may not be desirable.

Verbosity of Merge Analysis. By default, *findmerge*:

- Silently skips elements that do not require a merge.
- Issues a warning message if your view does not select any version of an element, but the view specified with `–ftag` does. (This occurs only when all elements of a VOB are being considered, with `–all` or `–avobs`, and a new element has been created in the "from" view.)

The following options override this behavior.

–why·not For each element that does not require a merge, displays a message explaining the reason.

–vis·ible Suppresses the warning messages for elements that are not visible in the current view.

Actions to be Performed on the Selected Elements. *Default:* None.

–pri·nt [–l·ong | –s·hort | –nxn·ame]
Lists the names of the elements that require a merge. The default listing includes the version-IDs of the "to" and "from" versions, and that of the base contributor (common ancestor):

```
Needs Merge "Makefile" [to /main/7 from /main/br1/1 base /main/6]
```

Specifying `–short` reduces the listing to version-extended pathnames of the "to" and "from" versions:

```
Makefile@@/main/7 Makefile@@/main/br1/1
```

Specifying `–long` adds to the default listing a description (*describe* command output) of the "from" version:

```
Needs Merge "Makefile" [to /main/7 from /main/br1/1 base /main/6]
version "Makefile@@/main/br1/1"
  created 09-Nov-93.11:18:39 by Allison K. Pak (akp.user@neptune)
```

```
element type: text_file
predecessor version: /main/br1/0
```

Specifying –nxname reduces the listing to just the standard pathname of the element:

```
./Makefile
```

–exe·c *command-invocation*
–ok *command-invocation*

Runs the specified command for each selected element. With –ok, *findmerge* pauses for verification on each element, thus allowing you to process some elements and skip others. Like the ClearCase *find* command, *findmerge* sets the following variables in the specified command's environment:

CLEARCASE_PN	pathname of element
CLEARCASE_XN_SFX	extended naming symbol (default: @@)
CLEARCASE_ID_STR	version-ID of "to" version
CLEARCASE_XPN	version-extended pathname of "to" version
CLEARCASE_F_ID_STR	version-ID of "from" version
CLEARCASE_FXPN	version-extended pathname of "from" version
CLEARCASE_B_ID_STR	version-ID of base contributor version

–mer·ge
–okm·erge
–xme·rge
–okx·merge

Performs a character-oriented or graphical merge for each element that requires it. The "ok" variants pause for verification on each element, thus allowing you to process some elements and skip others.

Special Case: Specifying -merge -xmerge causes *findmerge* to perform a character-oriented merge in –abort mode; if the merge aborts (because it could not proceed completely automatically), then the interactive graphical merge tool is invoked.

Logging of Merge Analysis. *Default:* A line is written to a *merge log file* for each element that requires a merge. The log takes the form of a shell script that can be used to perform, at a later time, merges that are not completed automatically (see –print and –abort, for example). A pound sign character (#) at the beginning of a line indicates that the required merge was performed successfully. The log file's name is generated by *findmerge* and displayed when the command completes.

–log *pname* Creates *pname* as the merge log file, instead of selecting a name automatically. To suppress creation of a merge log file, use -log /dev/null.

Merge Options. If you have *findmerge* actually perform merges, you can specify the following options, which work exactly as they do in the *merge* command. (–abort and –qall are mutually exclusive.)

–abo·rt Cancels a merge if it is not completely automatic.

–qal·l Turns off automated merging.

–ser·ial Reports differences with each line containing output from one contributor, instead of in a side-by-side format.

Specifying Checkout Comments. *Default:* When *findmerge* checks out elements in order to perform merges, it prompts for a single checkout comment (like `checkout -cq`). You can override this behavior by specifying one of the standard comment options.

–c *checkout-comment*
–cq
–cqe
–nc Standard comment options.

EXAMPLES

- Compare a source file version in your current view to a version on another branch. Log the results of the comparision, but do not perform the merge. (If a merge is required, the log file stores a command that will perform the merge.)

```
% cleartool findmerge msg.c –fversion /main/rel2_bugfix/LATEST –print
Needs Merge "msg.c" [to /main/2 from /main/rel2_bugfix/1 base /main/1]
A 'findmerge' log has been written to "findmerge.log.04-Feb-94.10:01:23"
% cat findmerge.log.04–Feb–94.10:01:23
cleartool findmerge msg.c@@/main/2 -fver /main/rel2_bugfix/1 -log /dev/null -merge
```

- For the current directory subtree, compare all versions visible in the current view against the versions selected by another view. Print a list of versions that require merging, but do not perform the merge. For versions where no merge is required, explain why.

```
% cleartool findmerge . –ftag rel2_bugfix_view –whynot –print
No merge "./Makefile" [/main/3 descended from /main/2]
No merge "./cm_add.c" [element not visible in view rel2_bugfix_view]
No merge "./hello.c" [to /main/4 from version zero /main/rel2_bugfix/0]
  .
  .
  .
A 'findmerge' log has been written to "findmerge.log.04-Feb-94.11:00:59"
% cat findmerge.log.04–Feb–94.11:00:59
cleartool findmerge ./msg.c@@/main/2 -fver /main/rel2_bugfix/1 -log /dev/null -merge
```

- For the current directory subtree, compare versions visible in the current view against versions on another branch, and perform any required merges. The resulting log file annotates all successful merges with a **#** character.

```
% cleartool findmerge . –fversion /main/rel2_bugfix/LATEST –merge
Needs Merge "./util.c" [to /main/3 from /main/rel2_bugfix/2 base /main/rel2_bugfix/1]
Comment for all listed objects:
Merge from rel2_bugfix branch.
.
Checked out "util.c" from version "/main/3".
******************************
<<< file 1: /tmp/george_fig_hw/src/util.c@@/main/rel2_bugfix/1
>>> file 2: ./util.c@@/main/rel2_bugfix/2
```

```
>>> file 3: ./util.c
*******************************
---------[changed 7-8 file 1]----------|--------[changed to 7-12 file 3]-------
    if (user_env)                      |     if (user_env) {
      return user_env;                 |       if ( strcmp(user_env,"root") == +
  .
  .
  .
Moved contributor "./util.c" to "./util.c.contrib".
Output of merge is in "./util.c".
Recorded merge of "./util.c".
A 'findmerge' log has been written to "findmerge.log.24-Mar-94.13:23:05"
% cat findmerge.log.24–Mar–94.13:23:05
#cleartool findmerge ./util.c@@/main/3 -fver /main/rel2_bugfix/2 -log /dev/nu
ll -merge -c "Merge from rel2_bugfix branch."
```

- As in the previous command, merge from another branch. This time, if any merge cannot be completed automatically (two or more contributors modify the same line from the base contributor), invoke the graphical merge utility to complete the merge.

  ```
  % cleartool findmerge . –fversion /main/rel2_bugfix/LATEST –merge –xmerge
  ```

SEE ALSO

cleartool subcommands: find, merge, xmerge
query_language, version_selector, wildcards_ccase, xcleardiff
find(1)

NAME help – help on cleartool command usage

SYNOPSIS

h·elp [*command-name*]

command-name **–h·elp**

DESCRIPTION

Displays a usage message for all *cleartool* subcommands, or for one particular subcommand. You can also use *help* as a command option — for example:

```
% cleartool des –h
```

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying a Subcommand. *Default:* Displays syntax summaries for all *cleartool* subcommands, grouped by function (not alphabetically).

command-name **–h·elp**
h·elp *command-name*
Displays the syntax summary for one *cleartool* subcommand.

EXAMPLES

- Display a usage message for the *mkview* command.

```
% cleartool help mkview
Usage: mkview -tag view-tag [-tcomment tag-comment] [-tmode text-mode]
              [-region network-region] [-ln link-storage-to-dir-pname]
              [-host hostname -hpath host-stg-pname -gpath global-stg-pname]
              view-storage-pname
```

- Display a usage message for the *mkview* command using the –help option.

```
% cleartool mkview –help
Usage: mkview -tag view-tag [-tcomment tag-comment] [-tmode text-mode]
              [-region network-region] [-ln link-storage-to-dir-pname]
              [-host hostname -hpath host-stg-pname -gpath global-stg-pname]
              view-storage-pname
```

- Display a usage message for all *cleartool* commands, and redirect the output to a file for future reference.

```
% cleartool help > cleartool_cmd_summary
```

SEE ALSO

cleartool subcommands: man
clearcase

NAME ln – create VOB hard link or VOB symbolic link

SYNOPSIS

- Create one link:

  ln [**–s·link**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname link-pname*

- Create one or more links in a specified directory:

  ln [**–s·link**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* [*pname …*] *target-dir-pname*

DESCRIPTION

NOTE: A link can be created in a directory only if that directory is checked-out. *ln* automatically appends an appropriate line to the directory's checkout comment.

The ClearCase *ln* command is similar to UNIX *ln*(1). It can create a single link, or it can create multiple links in a specified directory.

You can use both UNIX *ln* and the ClearCase *ln* command to create links in a VOB directory. Links created with UNIX *ln* are view-private objects and, thus, are invisible to users not working in your view. Links created with the ClearCase *ln* command (*VOB links*) are cataloged in directory versions, in the same way as elements. A VOB link becomes visible to those using other views only when you checkin the directory in which you create the link.

There are two kinds of VOB links:

- A *VOB hard link* (created if you omit the `–slink` option) is an additional name for an existing element.
- A *VOB symbolic link* (created if you use the `–slink` option) is a separate, unversioned object. Its contents is a character string, the *link text*, in the form of a pathname. You can attach attributes and hyperlinks to a VOB symbolic link, but not version labels.

The two kinds of VOB links differ in the way *checkout*/*checkin* works:

- You cannot checkout a VOB symbolic link. To "revise" a VOB symbolic link, you can (1) checkout its directory, (2) remove the old VOB link, (3) create a new link with the same name, and (4) checkin the directory.
- Since a VOB hard link is just another name for an element, you *can* checkout the link (that is, checkout the element it names). When you do so, all the *other* names for the element will be listed by a ClearCase *ls* command as `checkedout but removed`. (The element is checked-out, but there are no view-private files with the other names.) The command `lscheckout -all` lists the checked-out element only once.

VOB symbolic links and VOB hard links can be renamed with *mv*, and deleted with *rmname*.

VOB HARD LINKS AND DIRECTORY MERGES

The ClearCase *merge* and *findmerge* commands can merge both file elements and directory elements. Merging versions of a directory element can involve creation of a hard link to a directory:

- Working on a subbranch, a user checks out directory *src*, then uses *mkdir* to create (sub)directory element *testing* within *src*.
- When the subbranch is merged back into the main branch, a hard link named *testing* is made in a main-branch version of *src*, referencing the directory element already cataloged in the subbranch version.

Most UNIX implementations do not allow creation of such hard links to directories. ClearCase allows it only in this "directory merge" context: the two links (both named *testing* in the example above) must occur in versions of the same directory element (*src* in the example above).

RECOVERING A REMOVED ELEMENT

You can use *ln* to "recover" an element that you mistakenly removed from a VOB directory with *rmname*. See the *rmname* manual page for details.

UNIX HARD LINKS AND DERIVED OBJECTS

You cannot make a VOB hard link to a derived object, but you can make additional UNIX hard links to one. See the *derived_object* manual page for a discussion.

CAUTION

As with UNIX *ln*, the form of this command that creates multiple links in a directory is intended primarily for the creation of hard links. When using it to create VOB symbolic links, be sure not to create links that point to themselves. For example:

```
% cleartool ln –slink hello.c hello.h util.c subd
Link created: "subd/hello.c".
Link created: "subd/hello.h".
Link created: "subd/util.c".

% cd subd ; ls –l
total 3
lrwxrwxrwx   1 jackson      dvt       7 Mar  3 13:20 hello.c -> hello.c
lrwxrwxrwx   1 jackson      dvt       7 Mar  3 13:20 hello.h -> hello.h
lrwxrwxrwx   1 jackson      dvt       6 Mar  3 13:20 util.c -> util.c
```

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* An error occurs if any of the following objects are locked: VOB. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Type of Link. *Default:* Creates one or more VOB hard links.

–s·link Creates VOB symbolic links.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the 'To' Part. *Default:* None.

pname … Each *pname* must be a standard or view-extended pathname:

- For VOB hard links, each *pname* must specify an existing element. The element must be in the same VOB as the link being created. (NOTE: You cannot create a VOB hard link to a VOB symbolic link.)
- For VOB symbolic links, each *pname* is a character string that becomes the link text.

Specifying the Name of the New Link. *Default:* None.

link-pname A pathname within the same VOB as *pname*, at which one new VOB hard link or VOB symbolic link is to be created. An error occurs if an object already exists at *link-pname*.

target-dir-pname
The pathname of an existing directory element in the same VOB as the *pname* argument(s). *ln* creates a new link in this directory for each preceding *pname* argument. See "Caution" above.

EXAMPLES

- Create a VOB hard link, *hw.c*, as another name for element *hello.c*.

```
% cleartool ln hello.c hw.c
Link created: "hw.c".
```

- Create a VOB symbolic link, *messages.c*, pointing to *msg.c*.

```
% cleartool ln –slink msg.c messages.c
Link created: "messages.c".
```

- Create a group of hard links in the *subd* directory for all `.h` files in the current working directory.

```
% cleartool ln *.h subd
Link created: "subd/hello.h".
Link created: "subd/msg.h".
Link created: "subd/util.h".
```

SEE ALSO

cleartool subcommands: catcr, checkout, describe, ls, lsdo, merge, mkbranch, mv, rmname
derived_object, profile_ccase, ln(1)

NAME lock – lock an object

SYNOPSIS

- Lock entire VOB:

  lock [**–nus·ers** *login-name*[,...] | **–obs·olete**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  –vob { *pname-in-vob* | *vob-storage-dir-pname* }

- Lock VOB storage pool:

  lock [**–rep·lace**] [**–nus·ers** *login-name*[,...] | **–obs·olete**]
  [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  [**–vob** *pname-in-vob*] **–poo·l** *pool-name* ...

- Lock element or branch:

  lock [**–rep·lace**] [**–nus·ers** *login-name*[,...] | **–obs·olete**]
  [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* ...

- Lock type object:

  lock [**–rep·lace**] [**–nus·ers** *login-name*[,...] | **–obs·olete**]
  [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  { **–elt·ype** | **–brt·ype** | **–att·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–rpt·ype** }
  [**–vob** *pname-in-vob*] *type-name* ...

DESCRIPTION

Creates a *lock* on an entire VOB, or on one or more file system objects, *type objects*, or *VOB storage pools*. A lock on an object disables ClearCase operations that modify the object; a lock has no effect on "read" operations, such as *lshistory*. (Exception: see "Cleartext Pool" below.) The following sections describes the several kinds of locks.

VOB Lock

Locking an entire VOB disables all "write" operations to that VOB. A typical application is locking a VOB to prevent it from being modified during backup.

Type Lock

In general, locking a type object disables:

- operations that create, delete, or modify instances of the type
- operations that delete or modify the type object itself (for example, renaming it)

The following sections describe how these general rules apply to the different kinds of type objects.

Element Type. If an element type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mkeltype -replace` command

- create an element of that type with *mkelem* or *mkdir*
- change an existing element to that type with *chtype*
- modify the element's version tree with *checkout*, *checkin*, or *mkbranch*

Branch Type. If a branch type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mkbrtype -replace` command
- create a branch of that type with *mkbranch*
- rename (that is, change the type of) an existing branch to that type with *chtype*
- modify the branch with *checkout* or *checkin*

You *can* create a subbranch at any version on a locked branch, using *mkbranch*. (Creating a subbranch does not modify the branch itself.)

Label Type. If a label type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mklbtype -replace` command
- attach or remove a version label of that type with *mklabel* or *rmlabel* (This includes moving a label from one version to another with `mklabel -replace`.)

Attribute Type. If a attribute type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mkattype -replace` command
- attach or remove an attribute of that type with *mkattr* or *rmattr* (This includes moving an attribute from one version to another with `mkattr -replace`.)

Hyperlink Type. If a hyperlink type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mkhltype -replace` command
- create or remove a hyperlink of that type with *mkhlink* or *rmhlink*

Trigger Type. If a trigger type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mktrtype -replace` command
- (if created with `mktrtype -element`) create or remove a trigger of that type with *mktrigger* or *rmtrigger*

In general, locking a trigger type does not inhibit triggers of that type from firing. Exception: trigger firing is inhibited if a trigger type created with `mktrtype -element -global` or `mktrtype -type` is made obsolete (using `lock -obsolete`).

Replica Type. If a replica type is locked, you cannot:

- use it in an *rmtype*, *rntype*, or `mkrptype -replace` command
- create or remove a VOB replica of that type with *mkreplica* or *rmvob*.

Storage Pool Lock

Locking a VOB storage pool inhibits commands that create or remove the pool's data containers. It also prevents the pool's scrubbing parameters from being modified with `mkpool -update`. The following sections describe how this principle applies to the different kinds of storage pools.

Source Pool. If a source storage pool is locked, you cannot:

- create an element that would be assigned to that pool, with *mkelem* or *mkdir* (A new element inherits its pool assignments from its parent directory element.)
- change an existing element's pool assignment to/from that pool, with *chpool*
- change an element's element type with *chtype*, if the change would require recreation of source data containers (for example, changing from type *file* to type *text_file*)
- *checkin* a new version of an element assigned to that pool
- create or remove a branch of an element assigned to that pool, with *mkbranch* or *rmbranch*
- remove a version of an element assigned to that pool, or remove the element itself, with *rmver* or *rmelem*

Derived Object Pool. If a derived object storage pool is locked:

- *clearmake* cannot *wink-in* a previously unshared derived object in a directory assigned to that pool (The invocation of *promote_server* to copy the data container from view-private storage to the derived object storage pool fails.)
- *scrubber* cannot remove data containers from the pool
- an *rmdo* command fails for a derived object whose data container is in that pool

Cleartext Pool. If a cleartext storage pool is locked:

- an attempt to read (for example, with *cat*) a version of an element assigned to that pool may fail. (It fails if a new cleartext data container for that version would have been created and cached in the cleartext pool.)

OBSOLETE OBJECTS

An object becomes *obsolete* if it is processed with a `lock -obsolete` command. An obsolete type object or obsolete storage pool is not only locked, but is also invisible to certain forms of the *lstype*, *lslock*, and *lspool* commands. For example, the command `lstype -lbtype` omits obsolete label types from its listing.

An obsolete VOB or obsolete file system object is no different from one with an ordinary lock.

You can change an object's status from obsolete to "just plain locked" by using a `lock -replace` command:

```
% cleartool lock -obsolete -brtype test_branch          (make a branch type obsolete)
Locked branch type "test_branch".
% cleartool lock -replace -brtype test_branch           (change the branch type to 'just locked')
```

Similarly, you can use a `lock -replace` command to make a locked object obsolete.

REMOVING LOCKS

The *unlock* command removes a lock from an object, re-enabling the previously prohibited operations.

PERMISSIONS AND LOCKS

Kind of Object	Users Permitted to Lock the Object
type object	type creator, VOB owner, root user
storage pool	VOB owner, root user
VOB	VOB owner, root user
element	element owner, VOB owner, root user
branch	branch creator, element owner, VOB owner, root user

Even if you have permission to execute this command, it fails if an entire-VOB lock has been placed on the VOB containing the object.

OPTIONS AND ARGUMENTS

Replacing an Existing Lock. *Default:* An error occurs if you attempt to lock an object that is already locked.

−rep·lace (cannot be used when locking an entire VOB) Uses a single "atomic" transaction to replace an existing lock with a new lock. (If you use two commands to *unlock* the object, then *lock* it again, there is a short interval during which the object is "unprotected".)

You can use this option to change a object's status from "just locked" to "obsolete".

Specifying the Degree of Locking. *Default:* Locks an object to all users, but does not make the object obsolete.

−obs·olete Locks an object for all users, and also makes it obsolete.

−nus·ers *login-name*[,...]
Allows the specified users to continue using the object, which becomes locked to all other users. The list of user names must be comma-separated, with no white space.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: −nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Kind of Object to be Locked. *Default:* The final argument(s) are assumed to be the names of elements and or branches. To lock another kind of object, you must use one of the following options to specify the kind.

−vob The argument (only one allowed) will specify an entire VOB. This must be the last option specified — see "Specifying the VOB" below.

–elt·ype The argument(s) will specify element type objects.
–brt·ype The argument(s) will specify branch type objects.
–att·ype The argument(s) will specify attribute type objects.
–hlt·ype The argument(s) will specify hyperlink type objects.
–lbt·ype The argument(s) will specify label type objects.
–trt·ype The argument(s) will specify trigger type objects.
–rpt·ype The argument(s) will specify replica type objects.
–poo·l The argument(s) will specify storage pools.

Specifying the VOB. *Default:* When locking type objects and storage pools: processes objects in the VOB containing the current working directory; when locking an entire VOB: no default — you must specify a VOB.

–vob *pname-in-vob*
The VOB to be locked, or whose type object(s) or storage pool(s) are to be locked. *pname-in-vob* can be any location within the VOB.

–vob *vob-storage-dir-pname*
This alternative form of the –vob option is valid only when locking an entire VOB. This is a convenience feature, enabling administrators to process entire-VOB locks without having to use a view.

Specifying the Objects. *Default:* None.

type-name …
pool-name …
pname … (mutually exclusive)
One or more names, specifying the objects to be locked. To lock an element, you can specify the element itself (for example, *foo.c@@*) or any of its versions (for example, *foo.c* or *foo.c@@/RLS1.3*). To lock a branch, use an extended pathname (for example, *foo.c@@/main/rel2_bugfix*).

EXAMPLES

- Lock three label types for all users.

```
% cleartool lock –lbtype REL1 REL1.1 REL2
Locked label type "REL1".
Locked label type "REL1.1".
Locked label type "REL2".
```

- Obsolete a branch type.

```
% cleartool lock –obsolete –brtype rel2_bugfix
Locked branch type "rel2_bugfix".
```

- Lock the VOB containing the current working directory.

```
% cleartool lock –vob .
Locked versioned object base "/usr/hw".
```

- Lock the *motif* branch for all users except *gomez* and *jackson*.

```
% cleartool lock –nusers gomez,jackson –brtype motif
Locked branch type "motif".
```

- Lock elements with a `.c` suffix for all users. Then, try to checkout one of the locked elements.

```
% cleartool lock *.c
Locked file element "hello.c".
Locked file element "msg.c".
Locked file element "util.c".
```

```
% cleartool checkout –nc msg.c
cleartool: Error: Lock on file element prevents operation "checkout".
cleartool: Error: Unable to check out "msg.c".
```

SEE ALSO

cleartool subcommands: lshistory, lslock, lspool, lstype, protect, unlock
profile_ccase, promote_server, scrubber

NAME ls – list VOB-resident objects and view-private objects in a directory

SYNOPSIS

ls [**–r·ecurse** | **–d·irectory**] [**–s·hort** | **–l·ong**] [**–vob·_only** | **–vie·w_only**]
[**–nxn·ame**] [**–vis·ible**] [*pname* …]

DESCRIPTION

Lists *MVFS objects*: those accessed at pathnames within VOB directories. An error occurs if you try to list a non-MVFS object.

Listing Format

By default, *ls* lists:

- the name of each element cataloged in the current directory, with the version-ID of the particular version selected by the view. Also included is the version selector part of the config spec rule that selects this version.
- the name of each view-private object in the current directory
- the name of each derived object visible in the view, along with its unique DO-ID

The listing for an element or derived object may also include an annotation that indicates an unusual or noteworthy state. For example, the listing for an element that has been checked out to your view identifies the version that was checked out:

```
hello.c@@/main/CHECKEDOUT from /main/4         Rule: CHECKEDOUT
```

The following annotations may also appear:

`eclipsed`
No version of the element is selected, because a view-private object with the same name exists in your view. Typical occurrence: you create a view-private file in your view, then an element with the same pathname is created in another view; in your view, a `ls -vob_only` shows the element to be *eclipsed*.

`eclipsed by checkout`
(appears only when you use the `–vob_only` option) No version from the element's version tree is selected, because the element has been checked out in this view, and a checked-out version always eclipses all checked-in versions.

`checkedout but eclipsed`
The element has been checked out in this view, but there is no `CHECKEDOUT` config spec rule; thus, the checked-out version is not visible in the view.

`checkedout but removed`
The element was checked out in this view, but the view-private file was subsequently removed. You might have removed the file with UNIX *rm*(1). ClearCase removes it (in effect) when you checkout a file with `checkout -out`, or when you checkout a DO version.

NOTE: If a file element has several names, by virtue of one or more VOB hard links, checking out the element under one name causes all the *other* names to be listed with this annotation. (The element is checked-out, but there are no view-private files with the other names.)

`no version selected`
The element is not selected by any config spec rule, *or* is selected by a `-none` config spec rule.

`error on reference`
The element is selected by a `-error` config spec rule.

`view-->vob hard link`
The object is a view-private (UNIX-level) hard link to an object in VOB storage.

`no config record`
The derived object's data container is still stored in the view, but the derived object in the VOB database (and, typically, its associated configuration record) have been deleted by *rmdo*. This can occur only in the view in which the derived object was originally built.

`disputed checkout`
The element is considered to be checked-out by the *view_server* but is not so indicated in the VOB database (or vice-versa). This can occur during the short interval in which a *checkin* or *checkout* command is in progress. The annotation should not appear if you enter the *ls* command again.

`removed with white out`
The derived object was winked-in (and is still referenced) by the current view, but it has been forcibly removed from the VOB database with *rmdo*. The derived object is not recoverable.

Elements Suppressed from the View

The listing includes elements selected with `-none` and `-error` config spec rules, and elements that are not selected by *any* config spec rule. Standard commands, such as *ls*(1), and *cat*(1), get `not found` errors when accessing such elements. You *can* specify such elements in commands that access the VOB database only, such as *describe*, *lsvtree*, and *mklabel*.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Report Format. *Default:* The default report format is described in "Listing Format" above.

–s·hort Restricts the listing of each entry to its version-extended pathname only.

–l·ong Annotates listing of each entry with a classification — *version*, *directory version*, *view private object*, *derived object*, *derived object version*, or *symbolic link* — and lists the config spec rule that matches the object.

–nxn·ame Lists simple pathnames instead of version-extended pathnames.

VOB/View Restriction. *Default:* The listing includes both objects in VOB storage and objects in view storage.

–vob·_only
Restricts the listing to objects in VOB storage only, including versions of elements and VOB links. This may also *add* some entries to the listing: those for the underlying elements that are *eclipsed* by checked-out versions.

–vie·w_only
Restricts the listing to view objects only: view-private files, directories, and links; checked-out versions; and *all* derived objects visible in the view.

NOTE: Derived objects visible in the view are listed by `–view_only` (and not `–vob_only`), regardless of whether or not they are (or ever have been) shared.

–vis·ible Restricts the listing to objects visible to the standard *ls* command.

Specifying the Objects to be Listed. *Default:* The current working directory (equivalent to specifying "." as the *pname* argument). If you don't specify any other options, all files and links in the current working directory are listed; all subdirectory entries are listed, but not the contents of these subdirectories.

pname … Restricts the listing to the specified files, directories, and/or links.

Handling of Directory Arguments. *Default:* For each *pname* that specifies a directory element, *ls* lists the contents of that directory, but not the contents of any of its subdirectories.

NOTE: This includes directories in ClearCase's version-extended namespace, which represent elements and their branches. For example, specifying *foo.c@@/main/bug403* as an argument lists the contents of that branch: all the versions on the branch.

–r·ecurse Includes a listing of the entire subtree below any subdirectory included in the top-level listing. VOB symbolic links are not traversed during the recursive descent.

–d·irectory Lists information on a directory itself, rather than its contents.

EXAMPLES

NOTE: In some examples, output is wrapped for clarity.

- List the VOB-resident objects and view-private objects in the current working directory.

```
% cleartool ls
Makefile@@/main/3                                Rule:  /main/LATEST
bug.report
cm_add.c@@/main/0                                Rule:  /main/LATEST
cm_fill.c@@/main/0                               Rule:  /main/LATEST
convolution.c@@/main/CHECKEDOUT from /main/0     Rule:  CHECKEDOUT
edge.sh
hello@@24-Mar.11:32.418
hello.c@@/main/CHECKEDOUT from /main/4           Rule:  CHECKEDOUT
hello.h@@/main/CHECKEDOUT from /main/2           Rule:  CHECKEDOUT
hello.o@@24-Mar.11:32.412
hw.c@@/main/4                                    Rule:  /main/LATEST
include@@/main/CHECKEDOUT                        Rule:  CHECKEDOUT
```

- Use the objects in the current working directory, with annotations.

```
% cleartool ls –long
version               Makefile@@/main/3         Rule: element * /main/LATEST
view private object   bug.report
version               cm_add.c@@/main/0         Rule: element * /main/LATEST
view derived object   hello@@24-Mar.11:32.418
version               hello.h@@/main/CHECKEDOUT from /main/2
                                                Rule: element * CHECKEDOUT
view derived object   hello.o@@24-Mar.11:32.412
```

```
directory version      include@@/main/CHECKEDOUT Rule: element * CHECKEDOUT
symbolic link          messages.c --> msg.c
version                msg.c@@/main/1            Rule: element * /main/LATEST
view private object    util.c.contrib
```

- List only the view-private objects in the current working directory.

```
% cleartool ls –view_only
bug.report
hello@@24-Mar.11:32.418
hello.c@@/main/CHECKEDOUT from /main/4             Rule:  CHECKEDOUT
hello.h@@/main/CHECKEDOUT from /main/2             Rule:  CHECKEDOUT
hello.o@@24-Mar.11:32.412
msg.o@@23-Mar.20:42.379
util.c@@/main/CHECKEDOUT from /main/4              Rule:  CHECKEDOUT
util.o@@24-Mar.11:32.415
```

- List the contents of the directory in extended namespace that corresponds to the *main* branch of element *util.c*.

```
% cleartool ls util.c@@/main
util.c@@/main/0
util.c@@/main/1
util.c@@/main/2
util.c@@/main/3
util.c@@/main/CHECKEDOUT
util.c@@/main/LATEST
util.c@@/main/REL2
util.c@@/main/REL3
util.c@@/main/rel2_bugfix
```

SEE ALSO

cleartool subcommands: checkout, lsprivate, lsvtree, uncheckout
config_spec

NAME lscheckout – list checkouts of an element

SYNOPSIS

lsc·heckout | lsco [–r·ecurse | –d·irectory | –all | –avo·bs | –are·plicas]
[–l·ong | –s·hort | –fmt *format-string* **]**
[–me | –use·r *login-name* **] [–cvi·ew]**
[–brt·ype *branch-type-name* **] [** *pname …* **]**

DESCRIPTION

Lists the *checkout records* (the "checkouts") for one or more elements. There are many controls for specifying the scope: which elements, directories, or VOBs; which user; which view; and so on.

Checkouts and VOB Hard Links

A file element can have several names, by virtue of one or more *VOB hard links*. Checking out such an element under one name causes all the names to be listed as checked–out. But the command `lscheckout -all` lists the checked-out element only once.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Elements. *Default:* The current working directory (equivalent to specifying "." as the *pname* argument). If you don't specify any options, *lscheckout* lists all checkouts of elements in the current directory, to any view. If the current directory is itself checked-out, this is indicated, also.

pname … One or more pathnames, specifying file elements and/or versions of directory elements. (A standard or view-extended pathname to a directory specifies the version selected by the view.)

- For each *pname* that specifies a file element, the listing includes that element's checkout event records.
- For each *pname* that specifies a version of a directory element, the listing includes checkout event records of elements cataloged in that directory version.

The following options modify the processing of the *pname* arguments.

–r·ecurse Lists the checkouts of elements in the entire subtree below any directory encountered. VOB symbolic links are not traversed during the recursive descent.

–d·irectory Lists the checkouts (if any) of a directory itself, rather than the checkouts of elements cataloged in it.

–all Lists all the checkouts in the VOB containing *pname*. If you don't specify any *pname* arguments, lists all checkouts in the VOB containing the current working directory.

–avo·bs Similar to `–all`, but includes checkouts in all VOBs active (mounted) on the local host. (If environment variable CLEARCASE_AVOBS is set to a colon-separated list of VOB-tags, this set of VOBs is used instead.)

–are·plicas Similar to `–all`, but includes checkouts in all *replicas* of the VOB containing *pname*.

Report Format. *Default:* The listing of a checkout event record looks like this:

```
31-Aug.20:19  drp  checkout version "ct+lscheckout.1" from /main/24 (reserved)
```

–l·ong Expands the listing to include the view to which the element is checked out.

–s·hort Restricts the listing to the pathnames of checked-out elements only.

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

Selecting Checkout Records to List. *Default:* The listing includes all checkouts for the specified elements: made in any view, made by any user.

–me Restricts the listing to your own checkouts.

–use·r *login-name*
Restricts the listing to checkouts made by the specified user.

–cvi·ew Restricts the listing to checkouts made in the current view, not in other views.

–brt·ype *branch-type-name* Restricts the listing to checkouts on branches of the specified type.

EXAMPLES

NOTE: In some examples, output is wrapped for clarity.

- List the checkouts in the current working directory.

```
% cleartool lscheckout
08-Dec.12:17  jackson  checkout version "hello.c" from /main/4 (reserved)
08-Dec.12:17  jackson  checkout version "hello.h" from /main/1 (unreserved)
  "modify local defines"
08-Dec.12:17  jackson  checkout version "msg.c"
  from /main/rel2_bugfix/0 (reserved)
```

- List only the names of elements checked out to the current view.

```
% cleartool lscheckout –short –cview
hello.c
hello.h
hw.c
include
```

- List the checkouts in all directories at or below the current directory.

```
% cleartool lscheckout –recurse
08-Dec.12:17   jackson    checkout version "hello.c"
       from /main/4 (reserved)
08-Dec.12:17   jackson    checkout version "hello.h"
       from /main/1 (unreserved)
  "modify local defines"
08-Dec.12:17   jackson    checkout version "msg.c"
       from /main/rel2_bugfix/0 (reserved)
08-Dec.12:17   jackson    checkout directory version "subd"
       from /main/1 (reserved)
08-Dec.12:17   jackson    checkout version "./subd/util.h"
```

```
        from /main/0 (reserved)
```

- List elements checked out by the user in all mounted VOBs.

  % cleartool lscheckout –avobs –me

```
08-Dec.12:17   jackson    checkout version "/usr/hw/src/hello.c"
        from /main/4 (reserved)
08-Dec.12:17   jackson    checkout version "/usr/hw/src/hello.h"
        from /main/1 (unreserved)
  "modify local defines"
08-Dec.12:17   jackson    checkout directory version "/usr/hw/release"
        from /main/0 (reserved)
08-Dec.12:17   jackson    checkout version "/usr/hw/src/msg.c"
        from /main/rel2_bugfix/0 (reserved)
08-Dec.12:17   jackson    checkout version "/usr/hw/src/util.h"
        from /main/0 (reserved)
```

SEE ALSO

cleartool subcommands: checkin, checkout, lsprivate, uncheckout
fmt_ccase

NAME lsdo – list derived objects created by clearmake or clearaudit

SYNOPSIS

lsdo [**–r·ecurse**] [**–me**] [**–zer·o**] [**–l·ong** | **–s·hort** | **–fmt** *format-string*]
[**–sna·me** | **–sti·me**] [*pname* …]

DESCRIPTION

Lists information about one or more *derived objects* (DOs) in a VOB. Derived objects are created by the *clearmake* and *clearaudit* utilities. By default, *lsdo* lists all derived objects built at a given pathname, no matter what view they were built in. Exceptions:

- Unshared derived objects that have a zero *reference count* are omitted unless you use the `–zero` option.
- *DO versions*, derived objects that have checked in as versions of elements, are never listed.

You can use *pname* arguments to restrict the listing to derived objects with particular pathnames, or to all the derived objects in particular directories. You can specify a derived object with a standard pathname, or with an extended name that includes a derived object's unique *DO-ID*.

lsdo lists derived objects without respect to which views (if any) reference them. At any given time, a view "sees" at most one derived object at a given pathname.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of Directory Arguments. *Default:* If any *pname* argument is a directory, the DOs in *pname* are listed, but not the DOs in any subdirectories of *pname*.

–r·ecurse Includes DOs in the entire subtree below any *pname* that is a directory (or the current working directory if you don't specify any *pname* arguments). VOB symbolic links are *not* traversed during the recursive descent into a directory.

Selection of Derived Objects. *Default: lsdo* lists DOs created by any user, but excludes DOs whose data containers no longer exist.

–me Restricts the listing to derived objects that you created.

–zer·o Includes in the listing unshared (that is, never-shared) derived objects with zero reference counts. Such objects cannot be candidates for *configuration lookup* and *wink-in*, because their data containers no longer exist.

Controlling Report Appearance. *Default:* Each DO's listing includes its extended name (including DO-ID) along with creation-related data: time, username, and hostname. For example:

```
11-Jun.12:00  akp  "hello.o@@11-Jun.12:00.554" on neptune
```

In a listing of several DOs, the entries are sorted by derived object name. Within a group of like-named DOs, the entries are sorted chronologically, most recent entry first. The `–long`, `–short`, and `–fmt` options are mutually exclusive; the `–sname` and `–stime` options are mutually exclusive.

–l·ong Expands the listing to include a DO's reference count and the views that reference it.

–s·hort Restricts the listing for a DO to its extended name (including DO-ID).

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

–sna·me (confirms the default) Sorts the listing by derived object name.

–sti·me Sorts *all* entries chronologically, most recent entry first.

Specifying the Derived Object(s). *Default:* Lists all derived objects created in the current working directory.

pname … Standard pathnames and/or DO-IDs:

- A directory name causes all derived objects built in that directory to be listed.
- A standard or view-extended pathname of a file causes all derived objects built under that name to be listed.
- A pathname that includes a unique DO-ID (for example, *conv.c@@19-Nov.21:28.127450*) specifies a particular derived object to be listed.

EXAMPLES

- List, in reverse chronological order, all derived objects that you have created in the current working directory.

```
% cleartool lsdo –stime –me –short
ctl@@14-May.15:18.339307
ctl_V.o@@14-May.15:18.339305
libcmd.a@@14-May.15:16.339302
libcmd_V.o@@14-May.15:16.339300
cmd_type.o@@14-May.15:15.339297
cmd_view.o@@14-May.15:15.339294
cmd_utl.o@@14-May.15:15.339291
cmd_trig.o@@14-May.15:14.339288
cmd_lh.o@@14-May.15:14.339285
```

- List information on a derived object, identified by its extended pathname.

```
% cleartool lsdo util.o@@08–Dec.12:06.231
08-Dec.12:06   "util.o@@08-Dec.12:06.231"
```

- List all derived objects created in the current working directory with file name *hello*. Use the long format, to show which views reference the DOs; include DOs that are not referenced by any view.

```
% cleartool lsdo –long –zero hello
08-Dec-92.12:06:19     Chuck Jackson (test user) (jackson.dvt@oxygen)
  create derived object "hello@@08-Dec.12:06.234"
  references: 1  => oxygen:/usr/vobstore/tut/old.vws
08-Dec-92.12:05:35     Chuck Jackson (test user) (jackson.dvt@oxygen)
  create derived object "hello@@08-Dec.12:05.143"
```

SEE ALSO

cleartool subcommands: diffcr, catcr, rmdo
clearaudit, clearmake, crontab_ccase, fmt_ccase

NAME lshistory – list event records for VOB-database objects

SYNOPSIS

- File system data history:

  lsh·istory [**–min·or**] [**–nco**] [**–l·ong** | **–s·hort** | **–fmt** *format-string*]
  [**–eve·ntid**] [**–sin·ce** *date-time*] [**–use·r** *login-name*]
  [**–r·ecurse** | **–d·irectory** | **–a·ll** | **–avo·bs**]
  [**–bra·nch** *branch-type*] [*pname …*]

- Hyperlink history:

  lsh·istory [**–min·or**] [**–l·ong** | **–s·hort** | **–fmt** *format-string*] [**–eve·ntid**]
  [**–sin·ce** *date-time*] [**–use·r** *login-name*] **–hli·nk** *hlink-selector …*

- Type history:

  lsh·istory [**–min·or**] [**–nco**] [**–l·ong** | **–s·hort** | **–fmt** *format-string*]
  [**–eve·ntid**] [**–sin·ce** *date-time*] [**–use·r** *login-name*]
  [**–vob** *pname-in-vob*]
  { **–elt·ype** | **–brt·ype** | **–att·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–rpt·ype** }
  type-name …

- Storage pool history:

  lsh·istory [**–min·or**] [**–nco**] [**–l·ong** | **–s·hort** | **–fmt** *format-string*]
  [**–eve·ntid**] [**–sin·ce** *date-time*] [**–use·r** *login-name*]
  [**–vob** *pname-in-vob*] **–poo·l** *pool-name …*

- VOB history:

  lsh·istory [**–l·ong** | **–s·hort** | **–fmt** *format-string*] [**–eve·ntid**]
  [**–sin·ce** *date-time*] [**–use·r** *login-name*] **–vob** *pname-in-vob*

- VOB replica history:

  lsh·istory [**–l·ong** | **–s·hort** | **–fmt** *format-string*] [**–eve·ntid**] [**–sin·ce** *date-time*]
  [**–use·r** *login-name*] **–vre·plica** [**–vob** *pname-in-vob*] *replica-name …*

DESCRIPTION

Lists event records in reverse-chronological order, describing ClearCase operations that have affected a VOB's data. There are several kinds of listing:

- **File system data history** — Lists events concerning elements, branches, versions, and VOB links. This includes records for creation and deletion of objects, and records for attaching and removal of annotations: version labels, attributes, and hyperlinks.

- **Hyperlink history** — Lists events concerning hyperlink objects: creation, deletion, attaching/removal of attributes.
- **Type history** — Lists events concerning type objects that have been defined in the VOB.
- **Storage pool history** — Lists events concerning the VOB's storage pools.
- **VOB history** — Lists events concerning the VOB object itself. This includes the deletion of type objects and elements from the VOB.
- **VOB replica history** — Lists events concerning a VOB replica, including synchronization updates.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Default: If you don't specify any objects to be listed, *lshistory* reports on events concerning file system objects in the current working directory. (This is equivalent to specifying "." as the lone *pname* argument.) The following sections describe how to produce a report on other file system objects, or on other kinds of objects.

File System Data History. Use the following to specify one or more file system objects for a history listing.

pname … One or more pathnames, specifying elements and/or VOB symbolic links whose history is to be listed.

–bra·nch *branch-type*
Restricts the report to events relating to branches of the specified type. NOTE: You cannot use a *pname* argument like *foo.c@@/main* to restrict the report in this way.

–r·ecurse Processes the entire subtree below any directory element encountered. VOB symbolic links are not traversed during the recursive descent.

–d·irectory Lists information on a directory element itself, rather than on its contents.

–a·ll Reports on all objects in the VOB containing *pname*: file system objects, type objects, and storage pools. If you omit *pname*, uses the VOB containing the current working directory.

–avo·bs Similar to `–all`, but includes all VOBs active (mounted) on the local host. (If environment variable CLEARCASE_AVOBS is set to a colon-separated list of VOB-tags, this set of VOBs is used instead.) If a VOB has multiple replicas, events from all the replicas are reported.

Hyperlink History. Use the following to specify one or more hyperlink objects for a history listing.

–hli·nk *hlink-selector …* One or more names of hyperlink objects, in this form:

hyperlink-type-name@hyperlink-ID[*@pname-in-vob*]

Hyperlinks are not file system objects — you cannot specify them with shell wildcards. The final component is required only for a hyperlink in another VOB. Examples:

```
DesignFor@598f
RelatesTo@58843@/vobs/monet
```

Type History. Use the following to specify one or more type objects for a history listing.

–elt·ype (element type)
–brt·ype (branch type)
–att·ype (attribute type)
–hlt·ype (hyperlink type)
–lbt·ype (label type)
–trt·ype (trigger type)
–rpt·ype (replica type)
(mutually exclusive — exactly one required) Reports on events that involve the type objects specified with *type-name* arguments.

type-name …
One or more names of type objects. The types must exist in the VOB containing the current working directory, unless you specify another VOB with –vob *pname-in-vob*.

Storage Pool History. Use the following to select one or more storage pool objects for a history listing.

–poo·l *pool-name …*
One or more names of VOB storage pools. The pools must belong to the VOB containing the current working directory, unless you specify another VOB with –vob *pname-in-vob*.

VOB History. Use the following to select one or more VOBs for a listing of events on the VOB itself.

–vob *pname-in-vob*
Specifies the VOB whose history is to be listed. *pname-in-vob* can be the pathname of any object within the VOB.

VOB Replica History. Use the following to select one or more VOB replicas for a listing of replication-specific events.

–vre·plica [**–vob** *pname-in-vob*] *replica-name …*
Specifies the VOB replica whose history is to be listed. *pname-in-vob* can be the pathname of any object within the VOB.

Selecting Events for the Specified Objects. *Default:* The report includes all the "major" events in the entire histories of the selected objects.

–min·or Includes less important events in the listing: attaching of attributes, version labels, and so on. For type objects and storage pools, minor events include rename operations and changes to pool parameters (`mkpool -update`).

–sin·ce *date-time*
Restricts the report to events recorded since the specified date-time. The *date-time* argument can have any of the following formats:

date-time	:=	*date***.***time* | *date* | *time* | **now**
date	:=	*day-of-week* | *long-date*
day-of-week	:=	**today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date	:=	*d*[*d*]–*month*[–[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**

time := *h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

Examples:

```
22-November-1990
sunday
yesterday.16:00
8-jun
13:00
today
```

–nco Excludes `checkout version` events (the ones listed by the *lscheckout* command).

–use·r *login-name*
Restricts the report to events recorded for commands entered by the specified user.

Report Format. Default report format for an element:

```
02-Feb.10:51   scd       create version "msg.c@@/main/rel2_bugfix/1"
  "Version for branch creation test"
02-Feb.10:51   scd       create version "msg.c@@/main/rel2_bugfix/0"
02-Feb.10:51   scd       create branch "msg.c@@/main/rel2_bugfix"
  .
  .
  .
01-Feb.16:17   scd       create file element "msg.c@@"
```

Default report format for a hyperlink:

```
cleartool lshistory -hlink Merge@535@/tmp/scd_reach_hw
08-Feb.11:25   scd       create hyperlink "Merge@535@/tmp/scd_reach_hw"
```

Default report format for a storage pool:

```
01-Feb.16:05   scd       create pool "cdft"
  "Predefined pool used to store cleartext versions."
```

–l·ong Expands the listing to include other object-specific information.

–s·hort Restricts the listing to names only: pathnames of file system objects, names of type objects, or names of storage pools.

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

–eve·ntid Displays a numerical *event-ID* on the line that precedes each event record (even if you use `–fmt`). You can change the comment assigned to an arbitrary event record by supplying an event-ID to the `chevent -event` command. Event-IDs remain valid until the VOB is reformatted with *reformatvob*.

EXAMPLES

- List the event history of an element.

```
% cleartool lshistory hello.c
08-Dec.12:05   jackson    import file element "hello.c@@"
20-May.15:41   cory       create version "hello.c@@/main/3" (REL2)
  "include name, home dir, and time in message
   KNOWN BUG: extra NL at end of time message"
07-May.08:34   akp        create version "hello.c@@/main/2" (REL1)
  "ANSI compatibility: declare return value type, make explicit return value
   also: clean up wording for The Boss"
04-May.13:35   akp        create version "hello.c@@/main/1"
  "first implementation"
04-May.13:35   akp        create version "hello.c@@/main/0"
04-May.13:35   akp        create branch "hello.c@@/main"
04-May.13:35   akp        create file element "hello.c@@"
```

- List the events for an element that occurred after March 20, 1993, at 3 PM. Include minor events in the listing, such as meta-data modifications.

```
% cleartool lshistory –minor –since 20–mar–93.15:00 hello.c
08-Dec.12:12   jackson    modify meta-data file element "hello.c@@"
  "CHMOD +r"
08-Dec.12:05   jackson    import file element "hello.c@@"
20-May.17:35   cory       modify meta-data version "hello.c@@/main/3" (REL2)
  "Added label "REL2"."
20-May.15:41   cory       create version "hello.c@@/main/3" (REL2)
  "include name, home dir, and time in message
   KNOWN BUG: extra NL at end of time message"
15-May.14:46   ross       modify meta-data version "hello.c@@/main/2" (REL1)
  "Added label "REL1"."
07-May.08:34   akp        create version "hello.c@@/main/2" (REL1)
  "ANSI compatibility: declare return value type, make explicit return value
   also: clean up wording for The Boss"
04-May.13:35   akp        create version "hello.c@@/main/1"
  "first implementation"
04-May.13:35   akp        create version "hello.c@@/main/0"
04-May.13:35   akp        create branch "hello.c@@/main"
04-May.13:35   akp        create file element "hello.c@@"
  "first implementation"
```

- List the history of a label type, using the long format.

```
% cleartool lshistory –lbtype –long REL1
08-Jan-94.12:05:43     Chuck Jackson (test user) (jackson.dvt@oxygen)
  import label type "REL1"
15-Apr-94.14:45:00     ross.devt@neptune
  create label type "REL1"
  "create label for Release 1 of "hello world" program"
```

- For all elements in the current working directory, list events involving the *rel2_bugfix* branch.

```
% cleartool lshistory –branch rel2_bugfix
24-Mar.12:45   jackson        create version "msg.c@@/main/rel2_bugfix/0"
24-Mar.12:45   jackson        create branch "msg.c@@/main/rel2_bugfix"
  "release 2 bugfixes"
23-Mar.20:40   jackson        create version "util.c@@/main/rel2_bugfix/1"
  "fix bug: extra NL in time string"
```

```
23-Mar.20:39   jackson        create version "util.c@@/main/rel2_bugfix/0"
23-Mar.20:39   jackson        create branch "util.c@@/main/rel2_bugfix"
```

- List the history of the VOB object itself for the current VOB.

```
% cleartool lsh –vob .
10-Dec.08:01   gomez      unlock versioned object base "/home/gomez/personal"
09-Dec.15:48   gomez      lock versioned object base "/home/gomez/personal"
  "Locked for all users."
02-Oct.19:46   gomez      create versioned object base "/home/gomez/personal"
  "gomez's personal vob"
```

SEE ALSO

cleartool subcommands: describe, lscheckout, lspool, lstype, lsvtree, xlsvtree, chevent xclearcase, events_ccase, fmt_ccase

NAME lslock – list locks on objects

SYNOPSIS

- List locks on development data:

  lslock [**–s·hort** | **–l·ong** | **–fmt** *format-string*] [**–obs·olete**] [**–a·ll**] [*pname …*]

- List locks on type objects:

  lslock [**–s·hort** | **–l·ong** | **–fmt** *format-string*] [**–obs·olete**]
  [**–vob** *pname-in-vob*]
  { **–elt·ype** | **–brt·ype** | **–att·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–rpt·ype** }
  type-name …

- List locks on VOB storage pools:

  lslock [**–s·hort** | **–l·ong** | **–fmt** *format-string*] [**–obs·olete**]
  [**–vob** *pname-in-vob*] **–poo·l** *pool-name …*

- List the lock on an entire VOB:

  lslock [**–s·hort** | **–l·ong** | **–fmt** *format-string*] [**–obs·olete**]
  –vob { *pname-in-vob* | *vob-storage-dir-pname* }

DESCRIPTION

Lists *locks* that have been placed on one or more VOB-database objects (with the *lock* command). The listing can include all the locks created within a VOB, or just a particular set of locks:

- locks on elements or branches
- locks on type objects
- locks on VOB storage pools
- the lock on the VOB object itself

Obsolete Type Objects

Type objects can be rendered *obsolete* with the `lock -obsolete -xxtype` command. *lslock* lists an obsolete type object only if (1) you specify its name with a *type-name* argument, or (2) you use the `–obsolete` option.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Locked Objects. *Default:* Lists all the locks created in the VOB containing the current working directory.

Development data locks — use the following to list locks on elements and/or branches.

pname … One or more pathnames, each of which specifies an element or branch:

```
foo.c                          (element 'foo.c')
foo.c@@                        (element 'foo.c')
foo.c@@/main/bugfix            (branch of element 'foo.c')
```

(Versions cannot be locked; a pathname to a version references the element object.) Using *pname* arguments restricts the listing to locks on those particular objects (but see the `-all` description below).

NOTE: Specifying an element lists only the lock on the element itself, not on any of its branches.

-a·ll For each *pname* argument, lists all locks in the VOB containing *pname*. Has no effect if you don't specify any *pname* argument (since the default is to list all locks in the current VOB).

Type Object Locks — use the following to list locks on type objects:

-elt·ype *type-name …*
-brt·ype *type-name …*
-att·ype *type-name …*
-hlt·ype *type-name …*
-lbt·ype *type-name …*
-trt·ype *type-name …*
-rpt·ype *type-name …*
(mutually exclusive) Lists locks on the type objects specified by the *type-name* argument(s). The types must exist in the VOB containing the current working directory, unless you specify another VOB with `-vob` *pname-in-vob*.

-vob *pname-in-vob*
Specifies the VOB whose locks on type objects are to be listed. *pname-in-vob* can be the pathname of any object within the VOB.

Storage Pool Locks — use the following to list locks on VOB storage pools:

-poo·l *pool-name …*
One or more names of VOB storage pools. The pools must belong to the VOB containing the current working directory, unless you specify another VOB with `-vob` *pname-in-vob*.

-vob *pname-in-vob*
Specifies the VOB whose pool locks are to be listed. *pname-in-vob* can be the pathname of any object within the VOB.

VOB Locks — use the following to list the lock on an entire VOB:

–vob *pname-in-vob*
Specifies the VOB whose lock is to be listed. *pname-in-vob* can be the pathname of any object within the VOB.

–vob *vob-storage-dir-pname*
This alternative form of the –vob option is valid only listing VOB locks. This is a convenience feature, enabling administrators to list entire-VOB locks without having to use a view.

Report Format. *Default:* A lock listing looks like this:

```
01-Sep.08:42   drp       lock attribute type "AT2" (locked)
  "Locked for all users."
```

–l·ong Expands the listing with more time-specific and user-specific information.

–s·hort Restricts the listing to names of locked objects only.

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

Listing Obsolete Objects. *Default:* An obsolete object is not listed unless you specify it with a command-line argument.

–obs·olete Includes obsolete objects in the listing. (Has no effect if you specify one or more objects with arguments.)

EXAMPLES

- List the locks on three label types.

```
% cleartool lslock –lbtype REL1 REL1.1 REL2
08-Dec.12:19   jackson    lock label type "REL1" (locked)
  "Locked for all users."
08-Dec.12:19   jackson    lock label type "REL1.1" (locked)
  "Locked for all users."
08-Dec.12:19   jackson    lock label type "REL2" (locked)
  "Locked for all users."
```

- List the lock on a particular branch of a particular element.

```
% cleartool lslock util.c@@/main/rel2_bugfix
08-Dec.12:19  jackson  lock branch "util.c@@/main/rel2_bugfix" (locked)
  "Locked for all users."
```

- List the entire-VOB lock on the current VOB, in long format.

```
% cleartool lslock –long –vob .
08-Dec-92.14:57:58     Chuck Jackson (test user) (jackson.dvt@oxygen)
  lock versioned object base "/usr/hw" (locked)
  "Locked for all users."
```

- List all locked objects (including the obsolete ones) in the current VOB.

```
% cleartool lslock –obsolete
08-Dec.12:18   jackson    lock file element
       "/usr/hw/src/hello.c@@" (locked)
  "Locked for all users."
```

```
08-Dec.12:19   jackson    lock label type "REL1" (locked)
  "Locked for all users."
08-Dec.12:19   jackson    lock label type "REL2" (locked)
  "Locked for all users."
08-Dec.12:18   jackson    lock branch type "motif" (locked)
  "Locked except for users: gomez jackson"
08-Dec.12:18   jackson    lock branch type "patch3" (obsolete)
  "Locked for all users (obsolete)."
08-Dec.12:18   jackson    lock file element
        "/usr/hw/src/convolution.c@@" (locked)
  "Locked for all users."
08-Dec.12:19   jackson    lock branch
        "/usr/hw/src/util.c@@/main/rel2_bugfix@@" (locked)
  "Locked for all users."
```

- List the locks on two of the current VOB's storage pools.

```
% cleartool lslock –pool staged cdft
08-Dec.12:19   jackson    lock pool "staged" (locked)
  "Locked for all users."
08-Dec.12:19   jackson    lock pool "cdft" (locked)
  "Locked for all users."
```

SEE ALSO

cleartool subcommands: lock, ls, lshistory, lspool, lstype, unlock
fmt_ccase

NAME lspool – list VOB storage pools

SYNOPSIS

lspool [**–s·hort** | **–l·ong** | **–fmt** *format-string*] [**–obs·olete**]
[**–vob** *pname-in-vob*] *pool-name …*

DESCRIPTION

Lists information about one or more *VOB storage pools*. This listing does not include the elements assigned to the pool; use the *find* command for this purpose. Example:

```
% cleartool find /vobs/include –element 'pool(src_pool_2)' –print
```

Obsolete Storage Pools

Storage pools can be rendered *obsolete* with the `lock -pool -obsolete` command. The obsolete/non-obsolete status of a pool affects some forms of this command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Listing Format. *Default:* A storage pool listing looks like this:

```
20-Nov-1993    drp        pool "cdft"
```

–l·ong Expands the listing to pool parameters and pathnames.

–s·hort Restricts the listing to pool names only.

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

Listing Obsolete Pools. *Default:* If you don't specify any *pool-name* argument, a VOB's obsolete pools are suppressed from the listing.

–obs·olete Includes obsolete pools in the listing when you don't specify any *pool-name* argument. Has no effect if you specify one or more *pool-name* arguments.

Specifying the Pools. *Default:* Lists all storage pools in the VOB containing the current working directory.

pool-name …
One or more names of storage pools to be listed. A pool is listed whether or not it is obsolete.

–vob *pname-in-vob*
The VOB whose storage pools are to be listed. *pname-in-vob* can be the pathname of any object within the VOB.

EXAMPLES

- List all storage pools for the VOB containing the current working directory.

```
% cleartool lspool
08-Dec.12:21   jackson    pool "c_pool"
  "pool for c source files"
15-Dec.09:34   jenny      pool "cdft"
  "Predefined pool used to store cleartext versions."
08-Dec.12:21   jackson    pool "cltxt2"
15-Dec.09:34   jenny      pool "ddft"
  "Predefined pool used to store derived objects."
08-Dec.12:21   jackson    pool "do1"
08-Dec.12:21   jackson    pool "my_ctpool"
  "alternate cleartext pool"
15-Dec.09:34   jenny      pool "sdft"
  "Predefined pool used to store versions."
08-Dec.12:19   jackson    pool "staged"
```

- List information about a particular storage pool, in long format.

```
% cleartool lspool –long do1
pool "do1"
 08-Dec-92.12:21:13 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  kind: derived pool
  pool storage global pathname "/net/oxygen/usr/vobstore/tut/tut.vbs/d/do1"
  maximum size: 10000  reclaim size: 8000  age: 168
```

- List a particular storage pool, verifying that it is obsolete.

```
% cleartool lspool –short cltxt2
cltxt2 (obsolete)
```

SEE ALSO

cleartool subcommands: chpool, lock, mkpool, unlock
fmt_ccase

NAME lsprivate – list objects in a view´s private storage area

SYNOPSIS

lsp·rivate [**–tag** *view-tag*] [**–vob** *pname-in-vob*] [**–l·ong** | **–s·hort**] [**–co**]

DESCRIPTION

Lists the file system objects that belong to a view:

- view-private files, links, and directories
- unshared derived objects
- shared derived objects that are cataloged in (visible through) the view — even though these objects are stored in a VOB storage pool
- checked-out versions of file elements

Except for the shared derived objects, all of these objects are stored in the view's private storage area.

NOTE: This command does not list checked-out directory elements, because such a checkout does not produce a view-private object. Use the *lscheckout* command to list directory checkouts.

The objects are listed with full pathnames (thus including the VOB-tag), one per line.

STRANDED VIEW-PRIVATE FILES

lsprivate sometimes lists a view-private file in a special way, because it has become *stranded*: it has no name in the VOB namespace, as currently constructed by your view. There are several possible causes and, hence, several actions you can take.

File Still Accessible Through Some Directory Version

The *lsprivate* listing for a file can include a version-extended pathname to some directory element:

```
/usr/hw/src@@/main/3/subdir1/canUCme
```

In this example, file *canUCme* is stranded because its parent directory, *subdir1*, does not appear in the view as it is currently configured; but the file could be accessed through version */main/3* of directory element *src*, which does contain an entry for *subdir1*. (Note that you cannot use this "pathname" to access the view-private object. A version-extended pathname can refer only to an element, branch, or version — not to a view-private file.)

To make such a stranded file visible again, you must make its parent directory visible, by reconfiguring the view (in this case, to select version */main/3* of directory element *src*).

VOB Is Inactive

If a VOB is not currently active on your host, all view-private files corresponding to that VOB are temporarily stranded. *lsprivate* displays a warning message and prepends a # character to pathnames within that VOB:

```
cleartool: Warning: VOB not mounted: "/usr/hw"
  VOB UUID is 1127d379.428211cd.b3fa.08:00:69:06:af:65
  .
  .
  .
#/usr/hw/src/.cmake.state
#/usr/hw/src/findmerge.log.18-Mar-94.13:43:27
#/usr/hw/src/hello
```

```
#/usr/hw/src/hello.o
  ⋮
```

Reactivating the VOB on your host will restore *lsprivate* command output to normal for pathnames within that VOB.

VOB Is Inaccessible

If a VOB has been unregistered, all view-private files corresponding to that VOB are temporarily stranded; if the VOB has been deleted, the view-private files are permanently stranded. *lsprivate* cannot distinguish these cases; it may guess at the VOB's probable VOB-tag, but it lists the view-private files with an `Unavailable-VOB` prefix:

```
cleartool: Error: Unable to get VOB object registry information for
    replica uuid "1127d379.428211cd.b3fa.08:00:69:06:af:65".
cleartool: Warning: VOB is unavailable -- using name: "<Unavailable-VOB-1>".
  If it has been deleted use 'recoverview -vob <uuid>'
  VOB UUID is 1127d379.428211cd.b3fa.08:00:69:06:af:65
  Last known location of storage is phobos:/usr/people/david/tut/tut.vbs
#<Unavailable-VOB-1>/<DIR-3587d464.428211cd.b40c.08:00:69:06:af:65>/.cmake.state
#<Unavailable-VOB-1>/<DIR-3587d464.428211cd.b40c.08:00:69:06:af:65>/findmerge.log.18-Mar-94.13:43:27
```

Directory Element Has Been Deleted

If a directory element (or its entire VOB) has been deleted, all the corresponding view-private files are permanently stranded. They are listed with the VOB's UUID, as above, with no "cure" possible, except to use *recoverview* to move the files to the view's *lost+found* directory.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the View. *Default:* The current view is listed; a working directory view takes precedence over a set view.

–tag *view-tag*
: The view-tag of any registered view to which you have "read" access.

Selecting Objects to List. *Default:* All of the view's objects are listed.

–vob *pname-in-vob*
: Restricts the listing to objects for the specified VOB. *pname-in-vob* can be the pathname of any object within the VOB.

–co
: Restricts the listing to checked-out versions of file elements. Checked-out directory elements are *never* listed by *lsprivate*.

Listing Style. *Default:* Checked-out versions are annotated with `[checkedout]`.

–l·ong
: Lists objects in the style of `ls -long`.

–s·hort
: Lists pathnames only — no annotations.

EXAMPLES

- List the private objects in the view with view-tag *jackson_vu*, from the VOB identified by the pathname */usr/hw/src*.

```
% cleartool lsprivate –tag jackson_vu –vob /usr/hw/src
/usr/hw/src/bug.report
/usr/hw/src/convolution.c  [checkedout]
/usr/hw/src/edge.sh
/usr/hw/src/hello
/usr/hw/src/hello.c  [checkedout]
/usr/hw/src/hello.h  [checkedout]
/usr/hw/src/hello.o
/usr/hw/src/msg.o
/usr/hw/src/util.c  [checkedout]
/usr/hw/src/util.c.contrib
/usr/hw/src/util.c.contrib.1
/usr/hw/src/util.o
```

- List all checked-out versions of elements in the current view, from all VOBs.

```
% cleartool lsprivate –co
/usr/hw/src/convolution.c  [checkedout]
/usr/hw/src/hello.c  [checkedout]
/usr/hw/src/util.c  [checkedout]
/vobs/doc/PLAN/DocumentationProposal  [checkedout]
/vobs/doc/reference_man/test/attest.dat  [checkedout]
/vobs/doc/reference_man/test/testelem.c  [checkedout]
```

- List all elements in the current view, from all VOBs, using a long listing.

```
% cleartool lsprivate –long
view private object    /tmp/scd_reach/src/findmerge.log.04-Feb-94.10:01:01
view private object    /tmp/scd_reach/src/findmerge.log.04-Feb-94.11:00:59
version                /vobs/doc/reqs@@/main/CHECKEDOUT  from /main/33 Rule: element * CHECKEDOUT
version                /vobs/doc/specs@@/main/CHECKEDOUT from /main/7  Rule: element * CHECKEDOUT
```

SEE ALSO

cleartool subcommands: checkout, ls, lscheckout, reformatview

NAME lsreplica – list replicas of a VOB

SYNOPSIS

lsreplica [**–s·hort** | **–l·ong** | **–fmt·format**]
[**–tag** *vob-tag* | **–vob** { *pname-in-vob* | *vob-storage-dir-pname* }]
[**–rpt·ype** | *replica-name* …]

DESCRIPTION

Lists one or more of the *replicas* of a VOB. VOB replicas are created with the Atria *MultiSite* product.

EXAMPLES

- List all replicas of the VOB whose VOB-tag is */vobs/gvob_ech*.

```
% cleartool lsreplica –tag /vobs/gvob_ech
11-Mar.13:42   david      versioned object base replica "original"
11-Mar.13:45   david      versioned object base replica "second_rep"
```

SEE ALSO

cleartool subcommands: chevent, describe, lscheckout, lshistory, lslock, lsvob, lock, unlock

NAME lstype – list a VOB´s type objects

SYNOPSIS

lst·ype { **–elt·ype** | **–brt·ype** | **–lbt·ype** | **–att·ype** | **–hlt·ype** | **–trt·ype** | **–rpt·ype** }
[**–vob** *pname-in-vob*] [**–s·hort** | **–l·ong** | **–fmt** *format-string*] [**–obs·olete**]
[*type-name* …]

DESCRIPTION

Lists information about one or more of a VOB's *type objects*.

Obsolete Type Objects

Type objects can be rendered *obsolete* with the `lock -xxtype -obsolete` command. *lstype* lists an obsolete type object only if (1) you specify its name with a *type-name* argument, or (2) you use the `–obsolete` option.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Type Object. *Default:* None.

–elt·ype (element type)
–brt·ype (branch type)
–att·ype (attribute type)
–hlt·ype (hyperlink type)
–lbt·ype (label type)
–trt·ype (trigger type)
–rpt·ype (replica type)
(mutually exclusive) Specifies the kind type object to be listed. For example, `–attype` selects *attribute type* as the kind of object to be listed.

Listing Format. *Default:* A type object listing looks like this:

```
07-Nov-1993    sakai        element type "text_file"
```

–l·ong Expands the listing to include any type-specific parameters (for example, that a label type is "one-per-element"; that an element type inherited its type manager from the *text_file* supertype; and so on.)

–s·hort Restricts the listing to type names only.

–fmt *format-string*
Lists information using the specified format string. See the *fmt_ccase* manual page for details on using this report-writing facility.

Listing Obsolete Types. *Default:* If you don't specify any *type-name* argument, only the non-obsolete types of the specified kind are listed.

–obs·olete Includes obsolete type objects in the listing when you don't specify any individual type objects with *type-name* arguments. Has no effect if you specify one or more *type-name* arguments.

Specifying the VOB. *Default:* Lists type objects in the VOB that contains the current working directory.

–vob *pname-in-vob*
The VOB whose type object(s) are to be listed. *pname-in-vob* can be any location within the VOB.

Specifying Individual Type Objects. *Default:* All type objects of the specified kind (for example, all attribute type objects) are listed.

type-name …
One or more names of type objects. The listing will include only the named object(s).

EXAMPLES

- List all branch types defined in the VOB containing the current working directory.

```
% cleartool lstype –brtype
15-Dec.09:34   jenny      branch type "main"
  "Predefined branch type used to represent the main branch of elements."
08-Dec.12:12   jackson    branch type "motif"
  "motif development branch"
08-Dec.12:12   jackson    branch type "patch2"
08-Dec.12:12   jackson    branch type "patch3"
08-Dec.12:12   jackson    branch type "rel2_bugfix"
```

- List all label types defined in the current VOB. Use the short format, and include obsolete label types.

```
% cleartool lstype –lbtype –obsolete –short
CHECKEDOUT
LATEST
REL1 (obsolete)
REL2
REL3
V2.7.1 (obsolete)
```

Note that the listing includes the two predefined label types, *LATEST* and *CHECKEDOUT*.

- List information about a particular user-defined element type, in long format.

```
% cleartool lstype –long –eltype c_source
element type "c_source"
 08-Dec-92.12:12:38 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  type manager: text_file_delta (inherited from type "text_file")
  supertype: text_file
  meta-type of element: file element
```

- List information about a particular trigger type, in long format.

```
% cleartool lstype –trtype –long trig1
trigger type "trig1"
 08-Dec-92.12:14:08 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  element trigger
  pre-operation MODIFY_ELEM
  action: -exec checkcmt
```

- List information about a particular hyperlink type.

```
% cleartool lstype –long –hltype design_spec
hyperlink type "design_spec"
 08-Dec-92.12:13:31 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  "source to design document"
```

SEE ALSO

cleartool subcommands: describe, mkattype, mkeltype, mkhltype, mklbtype, mkbrtype, mktrtype, rmtype, rntype
fmt_ccase

NAME lsview – list view registry entries

SYNOPSIS

lsview [**–s·hort** | **–l·ong**] [**–hos·t** *hostname*] [**–reg·ion** *network-region*]
[*view-tag …* | **–sto·rage** *view-storage-dir-pname …*]

DESCRIPTION

Lists one or more ClearCase *views*. To be accessible to *cleartool* subcommands, including *lsview*, a view:

- must have an entry in the *view_object* registry file
- must have one or more entries in the *view_tag* registry file

These files, *view_object* and *view_tag*, constitute the *view registry* and are located in directory */usr/adm/atria/rgy* on the network's *registry server host*. The *registry_ccase* manual page describes the ClearCase registry files in detail.

Default Output

By default, *lsview* lists all views registered for the current network region, whether they are active or not. The default output line for each listed view looks like this:

```
* anneRel4    /net/host2/usr/viewstore/anneRel4.vws
```

The three output fields report:

- whether the view is active (`*`)
- the view-tag
- the view storage directory pathname

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Listing Format. *Default:* See "Default Output" above.

–l·ong Expands the listing to include all information stored in the *view registry* regarding the listed views. (See also the *registry_ccase* manual page.)

–s·hort Restricts the listing to view-tags only.

Specifying the Views. *Default:* List all views registered for the local network region.

–hos·t *hostname*
Confines the listing to views whose storage directories reside on host *hostname*.

–reg·ion *network-region*
Confines the view listing to include only the views registered for a particular network region. (The *mkview* and *mktag* commands have a `–region` option, which can be used to assign view-tags to specific network regions.) The *network-region* argument can include pattern-matching characters as described in *wildcards_ccase*. Single-quote the *network-region* argument, if it includes pattern-matching characters.

view-tag … Specifies a single view to be listed. The view must be registered, but it need not be active to be listed with *lsview*. The *view-tag* argument can include pattern-matching characters as described in *wildcards_ccase*. Single-quote any *view-tag* argument that includes pattern-matching characters.

–sto·rage *view-storage-dir-pname …*
One or more views, identified by full pathnames to their storage directories.

EXAMPLES

- List the views registered for the local network region.

```
% cleartool lsview
*  mainRel5    /net/host5/usr/viewstore/mainRel5.vws
   anneRel5    /net/host5/usr/viewstore/anneRel5.vws
*  anneTest    /net/host2/usr/anne/viewstore/anneTest.vws
   nordTest    /net/host2/usr/nord/nordtest.vws
*  nordRel5    /net/host4/usr/viewstore/nordRel5.vws
   nortRel4    /net/host8/usr/viewstore/nordRel4.vws
```

- List, using the long display format, the registry information for the view with view-tag *mainRel5*.

```
% cleartool lsview –long mainRel5
  Tag: mainRel5
    Global path: /net/host5/usr/viewstore/mainRel5.vws
    Server host: host5
    Region: main_headqtrs
    Active: YES
    View tag uuid:a9c1ba4d.853e11cc.a96b.08:00:69:06:05:d8
View on host: host5
View server access path: /usr/viewstore/mainRel5.vws
View uuid: a9c1ba4d.853e11cc.a96b.08:00:69:06:05:d8
```

- For a particular host, list the views whose view-tags match a wildcard pattern.

```
% cleartool lsview –host host4 '*anne*'
* anne_main /net/host4/usr/anne/views/anne_main.vws
  anne_rel2 /net/host4/usr/anne/views/anne_rel2.vws
```

SEE ALSO

cleartool subcommands: mkview, mktag, register, unregister
registry_ccase, view

NAME lsvob – list VOB registry entries

SYNOPSIS

lsvob [**–s·hort** | **–l·ong**] [**–hos·t** *hostname*] [**–reg·ion** *network-region*]
[*vob-tag* … | **–sto·rage** *vob-storage-dir-pname* …]

DESCRIPTION

Lists one or more VOBs. To be accessible to *cleartool* subcommands, including *lsvob*, a VOB must be *registered*. That is, it must have an entry in the *vob_object* file on the registry server host. In addition, each VOB typically has one or more entries in the *vob_tag* registry file, as you cannot mount, or even create, a VOB without assigning a tag to it. (See the *mkvob* manual page; the *registry_ccase* manual page describes the ClearCase registry files in detail.)

Default Output

By default, *lsvob* lists all VOBs registered for the current network region, whether they are mounted (active) or not. The default output line for each listed VOB looks like this:

```
* /vobs/src     /net/host2/usr/vobstore/src_vob    public
```

The four output fields report:

- whether the VOB is mounted (`*`)
- the VOB-tag
- the VOB storage directory pathname
- whether the VOB is public or private (see the *mkvob* manual page)

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Listing Format. *Default:* See "Default Output" above.

–l·ong Expands the listing to include all information stored in the *VOB registry* regarding the listed VOBs. (See the *registry_ccase* manual page.)

–s·hort Restricts the listing to VOB-tags only.

Specifying the VOBs. *Default:* List all VOBs registered for the local network region, both mounted and unmounted, public and private.

–hos·t *hostname*
Confines the listing to VOBs whose storage directories reside on host *hostname*.

–reg·ion *network-region*
Confines the VOB listing to include only the VOBs registered for one or more network regions. (The *mkvob* and *mktag* commands have a `–region` option, which can be used to assign VOB-tags to specific network regions.) The *network-region* argument can include pattern-matching characters as described in *wildcards_ccase*. Single-quote the *network-region* argument, if it includes pattern-matching characters.

vob-tag … Specifies one or more VOBs to be listed. A VOB must be registered, but it need not be mounted, to be listed with *lsvob*. The *vob-tag* argument can include pattern-matching characters as described in *wildcards_ccase*. Single-quote any *vob-tag* argument that includes pattern-matching characters.

–sto·rage *vob-storage-dir-pname …*
One or more VOBs, identified by full pathnames to their storage directories.

EXAMPLES

- List the VOBs registered for the local network region.

```
% cleartool lsvob
* /vobs/demo       /net/host5/usr/vobstore/demo_vob      public
* /vobs/src        /net/host2/usr/vobstore/src_vob       public
* /vobs/design     /net/host2/usr/vobstore/design_vob    public
  /vobs/doc        /net/host2/usr/vobstore/doc_vob       public
* /vobs/stage      /net/host4/usr/vobstore/stage_vob     public
  /vobs/bugvob     /net/host8/usr/anne/vobstore/bug_vob  private
```

- List, using the long display format, the registry information for the VOB with VOB-tag */vobs/vob12*. The output line `Active: YES` indicates that the VOB is currently mounted.

```
% cleartool lsvob –long /vobs/vob12
Tag: /vobs/vob12
  Global path: /net/host5/usr/vobstore/vob12.vbs
  Server host: host5
  Access: public
  Mount options: rw,soft
  Region: us_west
  Active: YES
  Vob tag replica uuid:cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3
Vob on host: host5
Vob server access path: /usr/vobstore/vob12.vbs
Vob family uuid:  aed00001.9d3e11ca.bc4c.00:01:53:00:e8:c3
Vob replica uuid: cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3
```

- For a particular host, list the VOBs whose VOB-tags match a wildcard pattern.

```
% cleartool lsvob –host host4 '*anne*'
* /usr/anne/vobs/test2 /net/host4/usr/anne/vobs/test2.vbs public
* /usr/anne/vobs/work  /net/host4/usr/anne/vobs/work.vbs  private
```

SEE ALSO

cleartool subcommands: mkvob, mount, umount, register, unregister, mktag
registry_ccase

NAME lsvtree – list version tree of an element

SYNOPSIS

lsvt·ree [**–nr·ecurse**] [**–a·ll**] [**–mer·ge**] [**–nco**] [**–s·hort**]
[**–bra·nch** *branch-pname*] *pname …*

DESCRIPTION

Lists part or all of the version tree of one or more elements. By default, the listing includes all branches of an element's version tree, but only certain versions on those branches. Command options control which branches, how many branches, and which versions are listed. You can also control the way version are annotated with version labels and merge arrows.

Graphical Version Tree Display

As an alternative to *lsvtree*'s character–based listing, the *xlsvtree* command provides an interface to the *xclearcase* utility, which displays version trees graphically.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Selecting the Starting Point. *Default:* Starts the version tree listing at an element's *main* branch.

–bra·nch *branch-pname*
Starts the version tree listing at the specified branch. You can also use an extended name as the *pname* argument (for example, *foo.c@@/main/bug405*) to start the listing at a particular branch.

Listing Subbranches. *Default:* Lists the entire subtree of the branch selected as the starting point.

–nr·ecurse Omits all subbranches from the listing, showing only versions on a single branch.

Selecting and Annotating Versions on a Branch. *Default:* For each branch included in the listing, these selected versions are listed: (1) checked-out versions (annotated with the view name) and their predecessors; (2) versions that are the LATEST on their branches; (3) versions that have been assigned version labels; (4) versions at which a subbranch was created. A version is annotated with up to five of its version labels; an ellipsis (…) indicates that the version has additional labels.

–a·ll Lists all versions on a branch, not just the selected versions; annotates each version with *all* of its version labels.

–mer·ge Includes all versions that are at the "from" end of one or more merge arrows (hyperlinks of type *Merge*). Annotations on each such version indicate the corresponding "to" objects.

–nco Excludes checked-out versions from the listing. The predecessor of a checked-out version is also excluded, unless there is another reason to include it (for example, it has a version label).

–s·hort Restricts the listing to version-extended pathnames. Version labels, merge annotations, and checkout annotations are omitted.

Specifying the Elements or Branches. *Default:* None.

pname … One or more pathnames, specifying elements or branches of elements. (Alternatively, use the –branch option to specify a branch of an element.)

EXAMPLES

- List selected versions from an element's version tree.

```
% cleartool lsvtree util.c
util.c@@/main
util.c@@/main/1 (REL2)
util.c@@/main/rel2_bugfix
util.c@@/main/rel2_bugfix/1
util.c@@/main/3 (REL3)
util.c@@/main/4
```

- List all versions on the *rel2_bugfix* branch of an element's version tree.

```
% cleartool lsvtree –branch /main/rel2_bugfix –all util.c
util.c@@/main/rel2_bugfix
util.c@@/main/rel2_bugfix/0
util.c@@/main/rel2_bugfix/1
```

SEE ALSO

cleartool subcommands: describe, lshistory, ls, lstype, xlsvtree
xclearcase

NAME man – display a ClearCase manual page

SYNOPSIS

man [*command_name*]

DESCRIPTION

Formats and displays the specified ClearCase on-line manual page. For *cleartool* subcommands, you can use any valid command abbreviation or alias. (You need not include the manual page source file's `ct+` prefix.) For example:

```
% cleartool man lscheckout          (full command name)
% cleartool man lsch                (abbreviation)
% cleartool man lsco                (alias)
% cleartool man ct+lscheckout       (actual filename)
% cleartool man ct+lsco             (incorrect: does not match actual filename)
```

With no arguments, *man* displays the *cleartool* overview manual page. To display an alphabetical listing of all ClearCase manual pages, use `cleartool man clearcase` or `cleartool man toc`.

USAGE OF MANPATH

ClearCase manual pages are stored in subdirectories of */usr/atria/doc/man*. The *man* subcommand modifies the environment to include a MANPATH variable set to this directory. It then executes the UNIX *man*(1) command in a subprocess. Thus, the shell from which you invoke *cleartool* need not have MANPATH set.

If, however, you wish to use UNIX *man* directly, without going through *cleartool*, be sure to include */usr/atria/doc/man* in your MANPATH. For example:

```
setenv MANPATH /usr/catman:/usr/man:/usr/atria/doc/man
```

Note that with UNIX *man*, you must match the manual page source file name:

```
% man ct+describe                  (correct)
% man describe                     (incorrect)
% man des                          (incorrect)
```

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Manual Page. *Default:* Displays the overview manual page for *cleartool*.

command_name

The name (or abbreviation, or alias) of a *cleartool* subcommand; or the name of any other ClearCase manual page.

EXAMPLES

- Display the manual page for the *mkview* command.

```
cleartool> man mkview
Reformatting page.  Wait...
  ⋮
```

- Display a Table of Contents for all *cleartool* commands.

```
% cleartool man toc
Reformatting page.  Wait...
  ⋮
```

- Display the *cleartool* overview manual page.

```
% cleartool man
Reformatting page.  Wait...
  ⋮
```

SEE ALSO

cleartool subcommands: help, apropos
clearcase, man(1), Permuted Index in this manual.

NAME merge – merge versions of a text-file element or a directory

SYNOPSIS

merge { **–to** *contrib-&-result-pname* | **–out** *output-pname* } [**–bas·e** *pname*]
[**–nda·ta** | **–nar·rows**] [**–rep·lace**] [**–abo·rt** | **–qal·l**]
[**–opt·ions** *pass-through-options*] [**–win·dow** | **–tin·y**]
[**–dif·f_format** | **–ser·ial_format** | **–col·umns** *n*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**]
[**–ins·ert** | **–del·ete**]
{ **–ver·sion** *contrib-version-selector …* | *contrib-pname …* }

DESCRIPTION

Calls an element-type-specific program (the *merge method*) to merge the contents of two or more files, or two or more directories. Typically the files are versions of the same file element; a directory merge *must* involve versions of the same directory element.

You can also perform a *subtractive merge*, which removes from a version the changes made in one or more of its predecessors.

Selection of a 'merge' Method

merge uses ClearCase's *type manager* mechanism to select a *merge* method very similarly to the way that the *diff* command selects a *compare* method. For details, see "Selection of a 'compare' Method" in the *diff* manual page.

FILE MERGE ALGORITHM

A merge is a straightforward extension of a file comparison. Instead of simply displaying the pairwise differences, the *merge* method analyzes the differences (sometimes with your help) and copies sections of text to the output file:

- Sections in which there are no differences among the contributors are automatically copied to the output file.
- If a difference section has exactly *one* contributor differing from the base file, the *merge* method automatically "accepts the change", and copies the contributor's modified section to the output file:

```
------------[changed 3-4]----|--------[changed to 3-4 file 2]---
now is the thyme             | now is the time
for all good men             | for all good people
                            -|-
*** Automatic: Applying CHANGE from file 2 [lines 3-4]
============
```

  (The `–qall` option turns off automatic acceptance of this kind of change.)

- For difference sections in which two or more contributors differ from the base file, the *merge* method senses the conflict, and prompts you to resolve it. It displays all the pairwise differences, and allows you to accept or reject each one for inclusion in the output file.

```
-------------[changed 10]----|---------[changed to 10 file 2]---
cent                         | sent
                            -|-
-------------[changed 10]----|---------[changed to 10 file 3]---
```

```
cent                         | scent
                             -|-
Do you want the CHANGE made in file 2?  [yes] no
Do you want the CHANGE made in file 3?  [yes] yes
Applying CHANGE from file 3 [line 10]
============
```

Be sure to verify that the changes you accept produce consistent merged output. For example, after performing a merge involving file *util.c*, you might compare files *util.c.contrib* (which contains its previous contents) and the new *util.c* (which contains the merge output).

Determination of the Base Contributor

If all the contributors are versions of the same element, *merge* determines the base contributor automatically. It examines the element's "merge-enhanced" version tree — the directed graph consisting of the actual version tree along with all the merge arrows created by previous merge operations. This examination reveals the relationships among versions from the standpoint of their contents ("which versions contributed their data to me?"), rather than from the standpoint of their creation order ("which versions were created before me?"). *merge* selects the "closest common ancestor" in this enhanced version tree to be the base contributor.

If no merges have been performed in the element, the actual common ancestor in the version tree is selected to be the base contributor. Figure 8 illustrates some common cases.

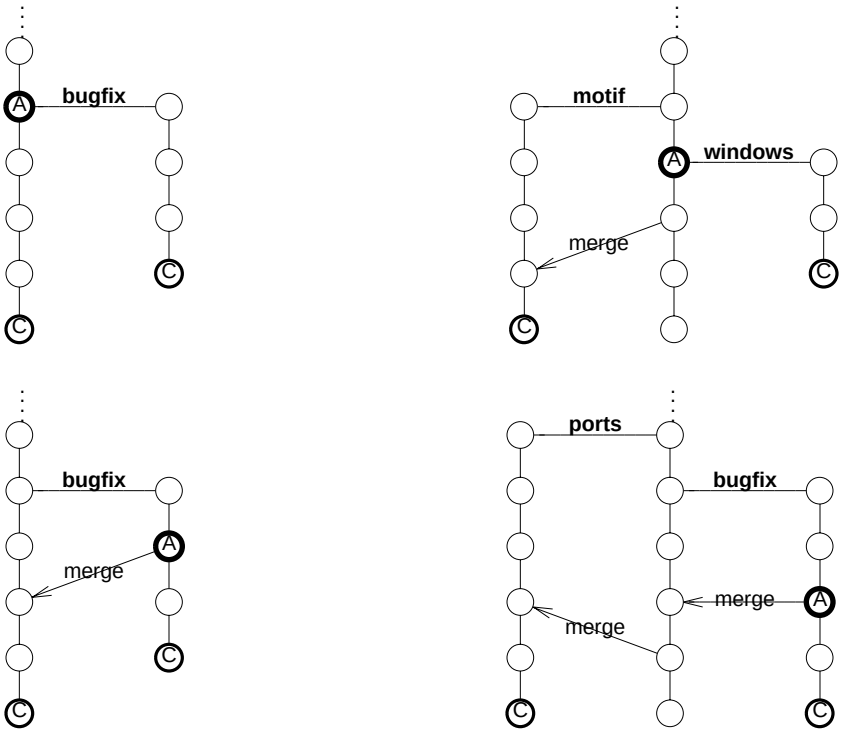


Figure 8. Determination of the Base Contributor for a Merge

If the contributors are *not* all versions of the same element, there is no base contributor. This means that you must resolve all discrepancies among the contributors; thus, the `-qall` option is enabled automatically.

Recording of Merge Arrows

Under certain circumstances, *cleartool* records the merge by creating one or more *merge arrows* (hyperlinks of type *Merge*):

- All contributor files must be versions of the same file element.
- One of the contributors must be a checked-out version, and you must specify this version as the *target* to be overwritten with the merge output (`-to` option).
- You must *not* use the `-narrows` option, which suppresses the creation of the merge arrow(s).

If all these conditions hold, *cleartool* draws an arrow from each contributor version (except the base contributor) to the target version.

Merge arrows can be viewed and traversed with *xclearcase*. The *find* command can locate versions with *Merge* hyperlinks. The *describe* command lists all of a version's hyperlinks, including merge arrows:

```
% cleartool describe util.c@@/main/3
```

```
version "util.c@@/main/3"
  :
  :
  Hyperlinks:
    Merge@278@/vob_3  /vob_3/src/util.c@@/main/rel2_bugfix/1
      -> /vob_3/src/util.c@@/main/3
```

NOTE: The `-ndata` option creates merge arrows without actually performing a merge.

DIRECTORY MERGE ALGORITHM

Each version of a ClearCase directory element *catalogs* (contains the names of) certain file elements, directory elements, and VOB symbolic links. *merge* can process two or more versions of the same directory element, producing a directory version that reflects the contents of all the contributors. The algorithm is similar to that for a file merge: *merge* compares a base contributor (common ancestor) version with each other contributor, producing a set of "pairwise differences". It applies these differences to the base contributor as automatically as possible, invoking a user interaction only when two or more of the pairwise differences are in conflict. (See the *diff* manual page for more on this algorithm.)

One of the directory versions — the merge target, specified with the `-to` option — must be checked out. (Typically, it is the version selected by your view.) *merge* updates the checked-out directory by adding, removing, and changing names of elements and/or links.

NOTE: Unlike a file merge, a directory merge does not leave behind a *.contrib* file, with the pre-merge contents of the target version. For this reason, we recommend that you use this procedure when merging directories:

1. Make sure that all contributor versions of the directory are checked in.
2. Checkout the target version of the directory.
3. Perform the directory merge immediately, without making any other changes to the checked-out version.

This procedure makes it easy to determine exactly what the merge accomplished: enter a `diff -predecessor` command on the checked-out version, which has just been updated by *merge*.

Using 'ln' to Implement a Merge

ClearCase implements directory merges using *VOB hard links*. You can use the ClearCase *ln* command to perform full or partial merges manually. See the *ln* manual page for details.

COMMON SCENARIOS

This section presents common scenarios for performing merges. See also the "Examples" section.

Case 1: Merging From a Branch

A common usage of this command is to combine the changes made on a subbranch (for example, a *bugfix* branch) of a file element with changes made on the *main* branch. The following illustration shows such a merge.

The target is *C1*, the checked-out version on the *main* branch, and the other contributor is *C2*, the latest version on a *bugfix* branch. *cleartool* automatically determines that version *B* is the common ancestor, to be used as the base file. The merged result replaces the contents of the target, *C1*, as shown in Figure 9.

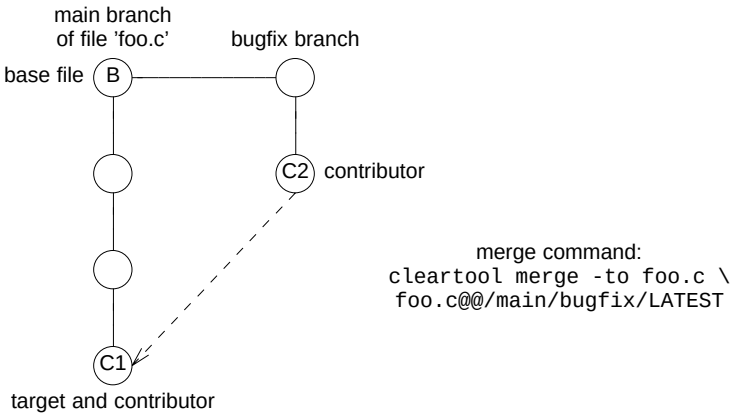


Figure 9. Merging From a Branch

Case 2: Merging into an Unreserved Checkout

Another common use of this command arises from ClearCase's *unreserved* checkout capability: you perform an unreserved checkout and edit the file, but someone else checks in a successor version ahead of you. You can check in your work only if you first merge with the version that "beat" you.

The next illustration shows a merge in which the target is *C1*, an unreserved, checked-out version that has lost the "race" to checkin. The other contributor is *C2*, the version that won the race. *cleartool* automatically determines that version *B* is the common ancestor, to be used as the base file. The result of the merge replaces the contents of the target, *C1*. *cleartool* will allow C1 to be checked in when it sees the merge arrow from C2 to C1, as illustrated in Figure 10.

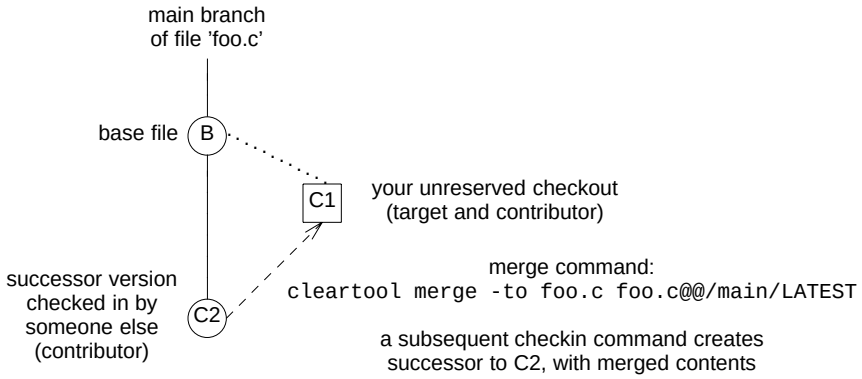


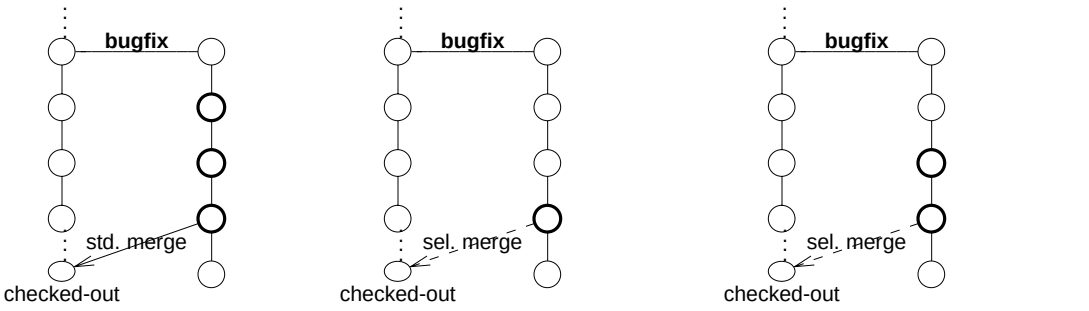
Figure 10. Merging into an Unreserved Checkout

SPECIAL MERGE SCENARIOS

merge has options that invoke special kinds of merges: selective and subtractive.

Selective Merges

By default, *merge* takes into account an entire, cumulative sequence of changes. For example, a merge from version */main/bugfix/4* back into the *main* branch involves the changes made in version 3, and also the changes made in versions 2 and 1 on that branch. In some cases, however, you may wish to incorporate just the changes made in one specific version (or a range of versions), disregarding the changes made in its predecessors. The `-insert` option implements a *selective merge* capability, as illustrated in Figure 11. In each merge, the heavy outlines indicate the versions on the *bugfix* branch whose changes are integrated back into the *main* branch.



standard merge:
```
cleartool merge \
-to foo.c \
-ver /main/bugfix/3
```

selective merge of one version:
```
cleartool merge \
-to foo.c \
-insert -ver /main/bugfix/3
```

selective merge of range of versions:
```
cleartool merge \
-insert -to foo.c \
-ver /main/bugfix/2 \
/main/bugfix/3
```

Figure 11. Selective Merge

In a selective merge, no merge arrow is created; such an arrow would indicate that *all* of a version's data has been merged, not just some of its data.

Subtractive Merges

The `-delete` option invokes a *subtractive merge*, which is the opposite of a selective merge:

- A selective merge *adds* to the checked-out version the changes made in one or more other versions.
- A subtractive merge *removes* from the checked-out version the changes made in one or more of its predecessors.

For example, to "undo" the changes made in versions 5 – 9 of file *foo.c*, while retaining all the changes made before version 5 and after version 9, you might issue this command:

```
cleartool merge -to foo.c -delete -ver /main/5 /main/9
```

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: (creation of 'merge arrows' only): element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: (creation of 'merge arrows' only): VOB, element type, element,

branch type, branch, hyperlink type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Destination of Merge Output. *Default:* None.

–to *contrib-&-result-pname*
Specifies a version of a file or directory element to be the *merge target*: one of the contributors to the merge, and also the location where the merged output is stored. *merge* proceeds as follows:

1. (file merge only) Preserves the target's current contents in view-private file *contrib-&-result-pname****.contrib***. The file name may get a *.n* extension, to prevent a name collision.
2. Stores the merged output in *contrib-&-result-pname*.

   You can suppress these data-manipulation steps by using `–ndata`; you *must* do so to avoid an error if the file is not checked-out:

   ```
   cleartool: Error: ...
   Only a checked out version can be modified to have the data
   resulting from the merge.
   ```

3. Creates a *merge arrow* (hyperlink of type *Merge*) from all other contributors to the checked-out version. You can suppress this step by using the `–narrows` option.

If the merge target cannot be overwritten, *merge* saves its work in view-private file *contrib-&-result-pname****.merge***. The file name may get a *.n* extension, to prevent a name collision.

–out *output-pname*
(file merge only) Specifies a view-private file or non-MVFS file to be the merge target. *output-pname* is not used as a contributor, and no merge arrows are created. Use this option to perform a merge that does not overwrite any of its contributors. An error occurs if *output-pname* already exists.

Specifying a Comment for the Merge Arrow. *Default:* Attaches a comment to each merge arrow (hyperlink of type *Merge*) according to your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Base Contributor. *Default:* Uses the procedure described in "Determination of the Base Contributor" above.

–bas·e *pname*
Specifies *pname* as the base contributor for the merge. You cannot use the `–version` option to specify this argument; use a version-extended pathname.

Suppressing Parts of the Merge Process. *Default: merge* stores its results in the location specified by `–to` or `–out`; with `–to`, it also creates merge arrows.

–nda·ta (use only with `–to`) Suppresses the merge, but creates the corresponding merge arrows.

An error occurs if you use `–ndata` along with `–out` — together, the two options leave *merge* with no work to do.

–nar·rows (for use with `–to`; invoked automatically by `–out`) Performs the merge, but suppresses the creation of merge arrows.

Replacing a Previous Merge. *Default:* An error occurs if a merge arrow is already attached to any version where *merge* would create one.

–rep·lace Allows creation of new merge arrows to replace existing ones.

Controlling User Interaction. *Default:* Works as automatically as possible, prompting you to make a choice only when two or more nonbase contributors differ from the base contributor.

–abo·rt Cancels the command instead of engaging in a user interaction; a merge takes place only if it is completely automatic. This command is useful in shell scripts that "batch" many merges (for example, all file elements in a directory) into a single procedure.

–qal·l Turns off automated merging. *merge* prompts you to make a choice every time a nonbase contributor differs from the base contributor. This option is turned on automatically if *merge* cannot determine a common ancestor (or other base contributor), and you do not use `–base`.

Invoking 'cleardiff' Options. *Default:* Does not pass any special options to the underlying *merge* method (implemented by the *cleardiff* utility for all predefined element types).

–opt·ions *pass-through-options*
Allows you to specify *merge* options that are not directly supported on the *merge* command line. For example, to invoke *cleardiff*'s `–quiet` option, use `cleartool merge –options -quiet`.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Using a Separate Window. *Default: merge* does its work in the current window.

–win·dow Performs the merge in a separate difference window. Output to this window is formatted as with `-columns 120`. When the merge finishes, type an operating system `INTR` character (typically, `<Ctrl-C>`) in the difference window to close it. The *merge* command does not terminate until you close the difference window.

–tin·y Same as `–window`, but uses a smaller font in a 165-character difference window.

Output Format. *Default:* Displays output in the format described in the ClearCase *diff* manual page.

–dif·f_format
Displays output in the same style as the standard *diff*(1) utility.

–ser·ial_format
Reports differences with each line containing output from one contributor, instead of in a side-by-side format.

–col·umns *n*
Establishes the overall width of side-by-side output. The default width is 80 — only the first 40 or so characters of corresponding difference lines appear. If *n* does not exceed the default width, this option is ignored.

Specifying Special Merges. *Default:* A standard merge is performed: all the differences between the base contributor and each nonbase contributor are taken into account.

–ins·ert Invokes a *selective merge* of the changes made in one or more versions. See the "Selective Merges" section above for a description. If you specify one contributor with `–version` or a *pname* argument, just that version's changes are merged. Specifying two contributors defines an inclusive range of versions; just the changes made in that range of versions are merged.

No merge arrow is created in a selective merge.

Restrictions: You must specify the target version with the `–to` option; this version must be checked-out. No version specified with `–version` or a *pname* argument can be a predecessor of the target version.

–del·ete Invokes a *subtractive merge* of the changes made in one or more versions. See the "Subtractive Merges" section above for a description. If you specify one contributor with `–version` or a *pname* argument, just that version's changes are removed. Specifying two contributors defines an inclusive range of versions; just the changes made in that range of versions are removed.

No merge arrow is created in a subtractive merge.

Restrictions: You must specify the target version with the `–to` option; this version must be checked-out. All versions specified with `–version` or a *pname* argument must be predecessors of the target version.

Specifying the Data to be Merged. *Default:* None.

–ver·sion *contrib-version-selector …*
(for use only if all contributors are versions of the same element) If you use the `–to` option to specify one contributor, you can specify the others with `–ver` followed by one or more version selectors. (See the *version_selector* manual page.)

contrib-pname …
One or more pathnames, indicating the objects to be merged: versions of file elements, versions of directory elements, or any other files. If you don't use `–to`, you must specify at least two *contrib-pname* arguments.

These two commands are equivalent:

```
% cleartool merge –to foo.c –version /main/bugfix/LATEST /main/3
% cleartool merge –to foo.c foo.c@@/main/bugfix/LATEST \
      foo.c@@/main/3
```

EXAMPLES

- Merge the version of file *util.c* in the current view with the most recent versions on the *rel2_bugfix* and *motif* branches; suppress the creation of merge arrows.

```
% cleartool merge –to util.c –narrows  \
      –version /main/rel2_bugfix/LATEST /main/motif/LATEST
```

- Merge the version of file *util.c*, in view *jackson_fix*, into version 3 on the *main* branch, placing the merged output in a temporary file.

```
% cleartool merge –out /tmp/proj.out util.c@@/main/3 \
      /view/jackson_fix/usr/hw/src/util.c
```

- Subtractive merge: remove the changes made in version 3 from file *util.c*.

```
% cleartool merge –to util.c –delete –version util.c@@/main/3
```

- Manually mimic the *findmerge* command functionality. Use the *find* command to determine which files need to be merged from the *rel2_bugfix* branch to the *main* branch, and invoke a script named *safemerge* to perform the merges. *safemerge* implements a "if-merge-aborts-use-xmerge" algorithm. It performs automatic merges with the *merge* command (determined by the `–abort` option), and merges that require operator intervention with the *xmerge* command. The *safemerge* script is shown immediately after the *find* command.

```
% cleartool find . –ele 'needs_merge(/main/rel2_bugfix,/main)' \
      –exec 'safemerge $CLEARCASE_PN rel2_bugfix'
```

safemerge script:

```
#!/bin/sh
#
#       arg1: element name
#       arg2: subbranch of /main
#
cleartool checkout -nc $1@@/main
cleartool merge -abort -to $1 $1@@/main/$2/LATEST > /dev/null 2>&1
if [ $? -ne 0 ] ; then
        cleartool xmerge -to $1 $1@@/main/$2/LATEST
fi
```

SEE ALSO

cleartool subcommands: describe, diff, find, findmerge, ln, mkhlink, rmmerge, xdiff, xmerge cleardiff, xcleardiff, xclearcase, profile_ccase, version_selector

NAME mkattr – attach attributes to objects

SYNOPSIS

- Attach attributes to specified objects:

  mkattr [**–rep·lace**] [**–r·ecurse**] [**–ver·sion** *version-selector*]
  [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  { *attribute-type-name value* | **–def·ault** *attribute-type-name* }
  [**–hli·nk**] *pname …*

- Attach attributes to versions listed in configuration record:

  mkattr [**–rep·lace**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  –con·fig *do-pname* [**–sel·ect** *do-leaf-pattern*] [**–ci**]
  [**–typ·e** { **f** | **d** } …] [**–nam·e** *tail-pattern*]
  { *attribute-type-name value* | **–def·ault** *attribute-type-name* }

DESCRIPTION

Prerequisite: An 'attribute type' object, created with 'mkattype', must already exist in the VOB(s) containing the objects to which you wish to attach attributes.

Attaches an *attribute* to one or more objects, each of which can be an element, a version, a branch, a VOB symbolic link, or a hyperlink. You can specify the objects themselves on the command line, or you can specify a particular derived object. In the latter case, *mkattr* attaches attributes to versions only — some or all the versions that were used to build that derived object.

An attribute is a *name/value* pair:

```
BugNum / 455                    (integer–valued attribute)
BenchMark / 12.9                (real–valued attribute)
ProjectID / "orange"            (string–valued attribute)
DueOn / 5-Jan                   (date–value attribute)
```

Restrictions on Attribute Usage

In several situations, attempting to attach a new attribute causes a *collision* with an existing attribute:

- You want to change the value of an existing attribute on an object.
- (if the attribute type was created with `mkattype -vpbranch`) An attribute is attached to a version, and you want to attach an attribute of the same type to another version on the same branch.
- (if the attribute type was created with `mkattype -vpelement`) An attribute is attached to a version, and you want to attach an attribute of the same type to any other version of the element.

A collision causes *mkattr* to fail and report an error, unless you use the `–replace` option, which first removes the existing attribute.

Referencing Objects by Their Attributes

ClearCase's *find* command can locate objects by their attributes. Examples:

`cleartool find . –element 'attype_sub(BugNum)' –print`
List all elements in the current working directory for which some version has been assigned a *BugNum* attribute.

`cleartool find util.c –version 'BugNum==4059' –print`
List the version of element *util.c* to which the attribute *BugNum* has been assigned with the value 4059.

More generally, queries written in the ClearCase *query language* can access objects using attribute types and attribute values. See the *query_language* manual page for details.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, attribute type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Moving an Attribute or Changing Its Value. *Default:* An error occurs if an attribute collision occurs (see "Restrictions on Attribute Usage" above).

–rep·lace Removes an existing attribute of the same type before attaching the new one, thus avoiding the collision. (No error occurs if a collision would not have occurred.)

Specifying the Attribute Type and Value. *Default:* None — you must specify an existing attribute type; you must also indicate a value, either directly or with the `–default` option.

attribute-type-name
An attribute type, previously created with *mkattype*. If the objects to be assigned attributes reside in different VOBs, the attribute type must exist in each VOB.

–def·ault If the attribute type was created with a default value (`mkattype -default`), you can use `–default` *attribute-type-name* to specify the name/value pair. An error occurs if the attribute type was not created with a default value.

value Specifies the attribute's value. The definition of the attribute type specifies the required form of this argument (for example, to an integer). It may also restrict the permissible values (for example, to values in the range 0–7).

Value Type	**Input Format**
integer	any integer that can be parsed by the *strtol*(2) system call
real	any real number that can be parsed by the *strtod*(2) system call
date	a date-time string in standard ClearCase format:

date-time := *date***.***time* | *date* | *time* | **now**
date := *day-of-week* | *long-date*
day-of-week := **today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date := *d*[*d*]–*month*[–[*yy*]*yy*]
month := **January** | ... | **December** | **Jan** | ... | **Dec**

time := *h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

string — any string in standard C-language string literal format. It must be enclosed in double quotes; it can include escape sequences: `\n`, `\t`, and so on.

NOTE: The double-quote (") character is special to both the *cleartool* command processor and the UNIX shells. Thus, you must escape or quote this character on the command line. These two commands are equivalent:

```
cleartool mkattr QAed '"TRUE"' hello.c
cleartool mkattr QAed \"TRUE\" hello.c
```

opaque — a "word" consisting of an even number of hex digits (for example, `04a58f` or `FFFB`). ClearCase stores the value as a byte sequence in a host-specific format.

Directly Specifying the Objects. The options and arguments in this section specify objects to be assigned attributes directly on the command line. Do not use these options and arguments when using a derived object to provide a list of versions to be assigned attributes.

pname … (required) One or more pathnames, indicating objects to be assigned attributes.

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A VOB-extended pathname specifies an element, branch, or version, independent of view.

Examples:

`foo.c`	*(version of 'foo.c' selected by current view)*
`/view/gamma/usr/project/src/foo.c`	*(version of 'foo.c' selected by another view)*
`foo.c@@/main/5`	*(version 5 on main branch of 'foo.c')*
`foo.c@@/REL3`	*(version of 'foo.c' with version label 'REL3')*
`foo.c@@`	*(the element 'foo.c')*
`foo.c@@/main`	*(the main branch of element 'foo.c')*

Use `–version` to override these interpretations of *pname*.

–ver·sion *version-selector*
For each *pname*, attaches the attribute to the version specified by *version-selector*. This option overrides both version–selection by the view and version–extended naming. See the *version_selector* manual page for syntax details.

–r·ecurse Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are *not* traversed during the recursive descent into the subtree.

NOTE: *mkattr* differs from some other commands in its default handling of directory element *pname* arguments: it assigns an attribute to the directory element itself; it does not assign attributes to the elements cataloged in the directory.

–hli·nk Indicates that the *pname* argument(s) name hyperlink objects, not file system objects. The hyperlink-specifier arguments take this form:

hyperlink-type-name@hyperlink-ID[*@pname-in-vob*]

Hyperlinks are not file system objects — you cannot specify them with shell wildcards. The final component is required only for a hyperlink in another VOB. Examples:

```
DesignFor@598f
RelatesTo@58843@/vobs/monet
```

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Using a Derived Object to Specify Versions. The options and arguments in this section specify versions to be assigned attributes by selecting them from the configuration record(s) associated with a particular derived object. Do not use these options when specifying objects to be assigned attributes directly on the command line.

–con·fig *do-pname*
(required) Specifies one derived object. A standard pathname or view-extended pathname specifies the DO that currently appears in a view. To specify a DO independent of view, use an extended name that includes a *DO-ID* (for example, `hello.o@@24-Mar.11:32.412`) or a version-extended pathname to a DO version.

With the exception of checked-out versions, *mkattr* attaches attributes to all the versions that would be included in a `catcr -flat` listing of that derived object. Note that this includes any DO created by the build and subsequently checked in as a DO version.

If the DO's configuration includes multiple versions of the same element, the attribute is attached only to the most recent version.

Use the following options to modify the list of versions to which attributes will be attached.

–sel·ect *do-leaf-pattern*
–ci
–nam·e *tail-pattern*
–typ·e { **f** | **d** } …
Modify the set of versions to be assigned attributes in the same way that these options modify a *catcr* listing. See the *catcr* manual page for details, along with the "Examples" section below.

EXAMPLES

- Create an attribute type named *BugNum*. Then, attach that attribute with the value 21 to the version of *util.c* that fixes bug 21.

```
% cleartool mkattype –nc –vtype integer BugNum
Created attribute type "BugNum".
% cleartool mkattr BugNum 21 util.c
Created attribute "BugNum" on "util.c@@/main/maintenance/3".
```

- Attach the *TESTED* attribute to the version of *hello.h* in the view, and assign it the value *TRUE*.

```
% cleartool mkattr TESTED '"TRUE"' hello.h
Created attribute "TESTED" on "hello.h@@/main/2".
```

- Update the value of the *TESTED* attribute on *hello.h* to `"FALSE"`. This example shows that to overwrite an existing attribute value, you must use the `–replace` option.

```
% cleartool mkattr –replace TESTED '"FALSE"' hello.h
Created attribute "TESTED" on "hello.h@@/main/2".
```

- Attach the *RESPONSIBLE* attribute to the *element* (not a particular version) *hello.c*.

```
% cleartool mkattr RESPONSIBLE '"Anne"' hello.c@@
Created attribute "RESPONSIBLE" on "hello.c@@".
```

- Attach the *TESTED_BY* attribute to the version of *util.c* in the view, assigning it the value of the USER environment variable as a double-quoted string. Using `\"` causes the shell to pass through the double-quote character instead of interpreting it. (Specifying the attribute value as `'"$USER"'` would not work, because the single-quotes suppress environment variable substitution.)

```
% cleartool mkattr TESTED_BY \"$USER\" util.c
Created attribute "TESTED_BY" on "util.c@@/main/5".
```

- Attach the *TESTED* attribute with its default value to each version that was used to build derived object *hello.o*. Note that the attribute is assigned both to files and to directories.

```
% cleartool mkattr –config hello.o –default TESTED
Created attribute "TESTED" on "/usr/hw/@@/main/1".
Created attribute "TESTED" on "/usr/hw/src@@/main/2".
Created attribute "TESTED" on "/usr/hw/src/hello.c@@/main/3".
Created attribute "TESTED" on "/usr/hw/src/hello.h@@/main/1".
```

- Attach the *TESTED* attribute with the value `"FALSE"` to those versions that were used to build *hello*, and whose pathnames match the `*.c` tail pattern.

```
% cleartool mkattr –config 'hello' –name '*.c' TESTED '"FALSE"'
Created attribute "TESTED" on "/usr/hw/src/hello.c@@/main/3".
Created attribute "TESTED" on "/usr/hw/src/util.c@@/main/1".
```

- Attach the *TESTED* attribute with the value `"TRUE"` to all versions in the VOB mounted at */src/lib* that were used to build *hello*. Use interactive mode to enable use of the ClearCase "..." wildcard.

```
% cleartool
cleartool> mkattr –config hello –name '/src/lib/...' TESTED '"TRUE"'
```

SEE ALSO

cleartool subcommands: catcr, describe, ln, lstype, mkattype, rmattr, rmtype, rntype
profile_ccase, query_language, version_selector

NAME mkattype – create an attribute type object

SYNOPSIS

mkattype [**–rep·lace**] [**–vpe·lement** | **–vpb·ranch** | **–vpv·ersion**]
[**–vob** *pname-in-vob*]
[**–vty·pe** { **integer** | **real** | **time** | **string** | **opaque** }]
[
{
[**–gt** *low-val* | **–ge** *low-val*] [**–lt** *high-val* | **–le** *high-val*]
|
–enu·m *value*[,...]
}
]
[**–def·ault** *default-val*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–oma·ster**] *type-name* ...

DESCRIPTION

Creates one or more *attribute types* for future use within a VOB. After creating an attribute type in a VOB, you can use *mkattr* to attach attributes of that type to several kinds of objects in that VOB: elements, branches, versions, and VOB symbolic links.

Attributes as Name/Value Pairs

An attribute is a *name/value* pair. When creating an attribute type, you must specify the kind of value (integer, string, and so on). You can also restrict the possible values to a particular list or range. Examples:

- Attributes of type *FUNC_TYPE* might be restricted to integer values in the range 1–5
- Attributes of type *QAed* might be restricted to the string values `"TRUE"` and `"FALSE"`.

Predefined Attribute Types

Each new VOB is created with two string-valued attributes types, named *HlinkFromText* and *HlinkToText*. When you enter a `mkhlink -ftext` command, the *from text* you specify is stored as an instance of *HlinkFromText* on the hyperlink object. Similarly, an *HlinkToText* attribute implements the *to text* of a hyperlink.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type creator (for –replace only), VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, attribute type (for –replace only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of Name Collisions. *Default:* An error occurs if an attribute type named *type-name* already exists in the VOB.

–rep·lace Replaces the existing definition of *type-name* with a new one. You must specify *all* options, even those that you wish to preserve from the old definition. Additional restrictions:

- If there are existing attributes of this type, you cannot change the `–vtype` value.
- If there are existing attributes of this type, you cannot replace a less restrictive `–vpelement`, `–vpbranch`, or `–vpversion` specification with a more restrictive one. (`–vpelement` is the most restrictive.)
- You cannot replace the predefined attribute types *HlinkFromText* and *HlinkToText*.
- When replacing an attribute type that was created with the `–omaster` option, you must use `–omaster` again; that is, you cannot convert an attribute type from shared to unshared.

Instance Restrictions. *Default:* In a given element, one attribute of the new type can be attached to each version, to each branch, and to the element itself. One attribute of the type can be attached to a VOB symbolic link.

–vpe·lement
Attributes of this type can be attached only to versions; and only one version of a given element can get an attribute of this type.

–vpb·ranch
Attributes of this type can be attached only to versions; and only one version *on each branch* of a given element can get an attribute of this type.

–vpv·ersion
Attributes of this type can be attached only to versions; within a given element, all versions can get an attribute of this type.

VOB Specification. *Default:* The attribute type is created in the VOB that contains the current working directory.

–vob *pname-in-vob*
Specifies the VOB in which to create the attribute type(s). *pname-in-vob* can be the pathname of any object within the VOB.

Specifying the Kind of Value. *Default:* One or more string-valued attribute types are created.

–vty·pe integer
Attributes of this type can be assigned integer values. You can use these options to restrict the possible values: `–gt`, `–ge`, `–lt`, `–le`, `–enum`.

–vty·pe real
Attributes of this type can be assigned floating-point values. You can use these options to restrict the possible values: `–gt`, `–ge`, `–lt`, `–le`, `–enum`.

–vty·pe time
Attributes of this type can be assigned values in the standard ClearCase date-time format. You can use these options to restrict the possible values: `–gt`, `–ge`, `–lt`, `–le`, `–enum`.

–vty·pe string
Attributes of this typ can be assigned character-string values. You can use the `–enum` option to restrict the possible values.

–vty·pe opaque
Attributes of this type can be assigned arbitrary byte sequences as values.

Restricting the Possible Values. *Default:* The values that can be assigned to attributes of the new type are unrestricted within the basic value type (*any* integer, *any* string, and so on). You can specify a list of permitted values, using `–enum`; alternatively, you can specify a range using `–gt` or `–ge` to specify the lower bound, and `–lt` or `–le` to specify the upper bound.

–gt *low-val* or **–ge** *low-val*
Lower bound of an *integer*, *real*, or *time* value. `–gt` means *greater than*. `–ge` means "greater than or equal to".

–lt *high-val* or **–le** *high-val*
Upper bound of an *integer*, *real*, or *time* value. `–lt` means "less than". `–le` means "less than or equal to".

–enu·m *value*[,…]
Comma-separated list (no white space allowed) of permitted values for any value type. See "Specifying the Attribute Type and Value" in the *mkattr* manual page for details on how to enter the various kinds of *value* arguments.

Specifying a Default Attribute Value. *Default:* Users will not be able to use `mkattr -default` to create an instance of this attribute type — they will have to specify an attribute value on the command line.

–def·ault *default-val*
Specifies a default attribute value; entering a `mkattr -default` command creates an attribute with the value *default-val*.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Attribute Types. *Default:* None.

type-name …
One or more names for the attribute types to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

Mastership of the Attribute Type. Attempts to attach or remove attributes of this type will succeed only in the *VOB replica* which is the current *master* of the attribute type. The VOB replica in which the new attribute type is created becomes its initial master.

–oma·ster Mastership of this attribute type will be ignored in *mkattr* and *rmattr* commands; instead, mastership of the object to which the attribute is being attached will determine whether the command succeeds.

EXAMPLES

- Create a string-valued attribute type named *Responsible*.

```
% cleartool mkattype –nc Responsible
Created attribute type "Responsible".
```

- Create an integer-valued attribute type named *Confidence_Level*, with a low value of 1 and a high value of 10. Restrict its use to one-per-branch.

```
% cleartool mkattype –nc –vpbranch –vtype integer –gt 0 –le 10 \
      Confidence_Level
Created attribute type "Confidence_Level".
```

- Create a string-valued attribute type named *QAed*, with an enumerated list of valid values.

```
% cleartool mkattype –nc –enum '"TRUE","FALSE","in progress"' QAed
Created attribute type "QAed".
```

- Create a time-valued attribute type named *QA_date*, with a default value of *today*. Provide a comment on the command line.

```
% cleartool mkattype –c "attribute for QA date" –vtype time \
      –default today QA_date
Created attribute type "QA_date".
```

- Create an enumerated attribute type, with a default value, called *Released*.

```
% cleartool mkattype –nc –enum '"TRUE","FALSE"' –default '"FALSE"' Released
Created attribute type "Released".
```

- Change the default value of an existing attribute type named *TESTED*. Provide a comment on the command line.

```
% cleartool mkattype –replace –default '"TRUE"' \
      –c "changing default value" TESTED
Replaced definition of attribute type "TESTED".
```

SEE ALSO

cleartool subcommands: describe, lstype, mkattr, mkeltype, rmattr, rntype
events_ccase, profile_ccase

NAME mkbranch – create a new branch in the version tree of an element

SYNOPSIS

mkbranch [**–ver·sion** *version-selector*] [**–nco**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
branch-type-name pname …

DESCRIPTION

Prerequisite: A 'branch type' object, created with 'mkbrtype', must already exist in the VOB(s) containing the specified elements.

Creates a new branch in the version trees of one or more elements. A *checkout* is performed on the new branch, unless you use the –nco option.

Auto-Make-Branch

The *checkout* command sometimes invokes *mkbranch* automatically. If the view's version of an element is selected by a config spec rule with a **–mkbranch** *branch-type* clause:

1. *checkout* first creates a branch of type *branch-type*.
2. It then checks out (version 0 on) the newly-created branch.

Similarly, entering a *mkbranch* command explicitly can invoke one or more *additional* branch-creation operations. See "Multiple-Level Auto-Make-Branch" in the *checkout* manual page.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, branch type, element, pool (non-directory elements only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Checkout of the New Branch. *Default:* The newly-created branch is checked out. Additional checkouts may ensue — see "Auto-Make-Branch" above.

–nco Suppresses automatic checkout of the branch.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –cqe). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Branch Type. *Default:* None.

branch-type-name
An existing branch type, previously created with *mkbrtype*. If the elements reside in different VOBs, the branch type must exist in each VOB.

Specifying the Branch Points. *Default:* None.

pname … One or more pathnames, indicating the versions at which branch are to be created.

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A version-extended pathname specifies a version, independent of view.

Use –version to override these interpretations of *pname*.

–ver·sion *version-selector*
For each *pname*, creates the branch at the version specified by *version-selector*. This option overrides both version–selection by the view and version–extended naming. See the *version_selector* manual page for syntax details.

EXAMPLES

- Create a branch type named *solaris*. Then, set a view that prefers versions on the *solaris* branch, and create a branch of that type in file *util.h*.

```
% cleartool mkbrtype –c "solaris development branch" solaris
Created branch type "solaris".
% cleartool setview solaris_port
% cleartool mkbranch –nc solaris util.h
Created branch "solaris" from "util.h" version "/main/1".
Checked out "util.h" from version "/main/solaris/0".
```

- Create a branch named *rel2_bugfix* off the version of *hello.c* in the view, and checkout the initial version on the branch.

```
% cleartool mkbranch –nc rel2_bugfix hello.c
Created branch "rel2_bugfix" from "hello.c" version "/main/4".
Checked out "hello.c" from version "/main/rel2_bugfix/0"
```

- Create a branch named *maintenance* off version */main/1* of file *util.c*. Do not checkout of the initial version on the branch.

```
% cleartool mkbranch –version /main/1 –nco –nc maintenance util.c
Created branch "maintenance" from "util.c" version "/main/1".
```

- Create a branch named *motif* off version */main/3* of file *hello.c*, and checkout the initial version on the branch. Use a version-extended pathname to specify the version.

```
% cleartool mkbranch –nc motif hello.c@@/main/3
Created branch "motif" from "hello.c" version "/main/3".
Checked out "hello.c" from version "/main/motif/0".
```

- For each file with a `.c` suffix, create a branch named *patch2* at the currently-selected version, but do not checkout the initial version on the new branch. Provide a comment on the command line.

```
% cleartool mkbranch –nco –c "release 2 code patches" patch2 *.c
Created branch "patch2" from "cm_add.c" version "/main/1".
Created branch "patch2" from "cm_fill.c" version "/main/3".
Created branch "patch2" from "msg.c" version "/main/2".
Created branch "patch2" from "util.c" version "/main/1".
```

SEE ALSO

cleartool subcommands: describe, lstype, mkbrtype, rmtype, rntype
profile_ccase, version_selector

NAME mkbrtype – create a branch type object

SYNOPSIS

mkbrtype [**–rep·lace**] [**–pel·ement**] [**–vob** *pname-in-vob*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *type-name* …

DESCRIPTION

Creates one or more *branch types* with the specified names for future use within a particular VOB. After creating a branch type in a VOB, you can create branches of that type in that VOB's elements, using *mkbranch*.

Instance Restrictions

ClearCase's version-extended naming scheme requires that a branch of a version tree have at most one subbranch of a given type. (If there were two *bugfix* subbranches of the *main* branch, then the version-extended pathname `foo.c@@/main/bugfix/3` would be ambiguous.) The `–pelement` option imposes a tighter restriction: only one branch of this type can be created in an element's entire version tree.

Recommended Naming Convention

A VOB cannot contain a branch type and a label type with the same name. For this reason, we strongly recommend that you adopt this convention:

- Make all letters in names of branch types lowercase (`a – z`).
- Make all letters in names of label types uppercase (`A – Z`).

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: (with –replace only): type creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, branch type (with –replace only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of Name Collisions. *Default:* An error occurs if a branch type named *type-name* already exists in the VOB.

–rep·lace Replaces the existing definition of *type-name* with a new one. You must specify *all* options, even those that you wish to preserve from the old definition. Additional restrictions:

- You cannot replace the predefined branch type *main*.
- If there are existing branches of this type, you cannot replace a less restrictive definition (no instance restriction) with a more restrictive definition (`–pelement` specified).

Instance Restrictions. *Default:* Multiple branches of the new type can be created in a given element. (NOTE: Each one must have a different branch as its "parent".)

–pel·ement
Only one branch of the new type can be created in a given element's version tree.

VOB Specification. *Default:* The branch type is created in the VOB that contains the current working directory.

–vob *pname-in-vob*
A pathname within any VOB. The branch type(s) will be created in that VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Branch Types. *Default:* None.

type-name …
One or more names for the branch types to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

See "Recommended Naming Convention" above.

EXAMPLES

- Create a branch type named *motif*, which can be used without restriction in an element's version tree. Provide a comment on the command line.

```
% cleartool mkbrtype –c "motif development branch" motif
Created branch type "motif".
```

- Create two branch types for working on program "patches", and a bugfix branch for "release 2". Restrict their use to one-per-element.

```
% cleartool mkbrtype –nc –pelement patch2 patch3 rel2_bugfix
Created branch type "patch2".
Created branch type "patch3".
Created branch type "rel2_bugfix".
```

- Change the usage restriction of an existing branch type such that it can be used only once per element. Provide a comment on the command line.

```
% cleartool mkbrtype –replace –pelement –c "change to one per element" motif
Replaced definition of branch type "motif".
```

SEE ALSO

cleartool subcommands: chtype, describe, lstype, mkbranch, rmtype, rntype
profile_ccase

NAME mkdir – create a directory element

SYNOPSIS

mkdir [**−nco**] [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *dir-pname …*

DESCRIPTION

NOTE: A new directory element can be created only if its parent directory is checked-out. *mkelem* automatically appends an appropriate line to the parent directory's checkout comment.

Creates one or more directory elements. (You can also use the standard UNIX *mkdir*(1) command, but that creates view-private directories, not elements.) Unless you specify the `–nco` ("no checkout") option, the new directory is checked out automatically. A directory element must be checked out before you can create elements and VOB links within it.

The `mkelem -elt directory` command is equivalent to this command.

The new directory element is assigned to the same storage pools (source, derived object, and cleartext) as its parent directory element. You can assign the directory to different pools with the *chpool* command.

You cannot create a directory element with the same name as an existing view-private file or directory.

Access Mode

In standard UNIX fashion, new directory elements are created with mode 777, as modified by your umask. The meanings of the "read", "write", and "execute" access permissions do not exactly their standard UNIX meanings, however. See the *protect* manual page for details.

Converting View-Private Directories

You cannot use this command to convert an existing view-private directory structure into directory and file elements. To accomplish this task, use the *clearcvt_unix* utility.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* An error occurs if any of the following objects are locked: VOB, element type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Checkout of the New Directory. *Default: mkelem* performs a *checkout* on the new directory element.

−nco Suppresses checkout of the new directory element.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Directory(s). *Default:* None.

dir-pname …

One or more pathnames, specifying directories to be created.

EXAMPLES

- Create a subdirectory named *subd*, and checkout the directory to the current view.

```
% cleartool mkdir –nc subd
Created directory element "subd".
Checked out "subd" from version "/main/0".
```

- Create a subdirectory named *release*, but do not check it out. Provide a comment on the command line.

```
% cleartool mkdir –nco –c "Storage directory for released files" release
Created directory element "release".
```

SEE ALSO

cleartool subcommands: cd, checkout, chpool, mkelem, mkpool, mv, protect, pwd, rmelem
clearcvt_unix, profile_ccase

NAME mkelem – create a file or directory element

SYNOPSIS

mkelem [**–elt·ype** *element-type-name*] [**–nco** | **–ci**]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *element-pname …*

DESCRIPTION

NOTE: A new element can be created in a directory only if that directory is checked-out. *mkelem* automatically appends an appropriate line to the directory's checkout comment.

Creates one or more new *elements*. For each one, *mkelem*:

1. determines an *element type* from the specified `–eltype` option, or by using the ClearCase file-typing facility (see below)
2. creates an element object with that element type in the appropriate VOB database
3. if you are using the `–ci` option to convert a view-private file to an element, uses the permissions of that file, including *set-UID* and/or *set-GID* bits; otherwise, sets the access mode of the new element to 444 (file element) or 777 (directory element), as modified by your current *umask*(1) setting
4. initializes the element's version tree by creating a single branch (named *main*), and a single, empty version (*version 0*) on that branch
5. either does nothing (`–nco` option), or creates version 1 by copying a view-private file (`–ci` option), or checks out the element to your view (default)
6. assigns the element to the same *source storage pool*, *cleartext storage pool*, and (for new directory elements only) *derived object storage pool* as its parent directory element

NOTE: Error messages appear if your config spec lacks a "/main/LATEST" rule. The *mkelem* command succeeds in creating version */main/0*; but since your view does not have a rule to select this version, the element cannot be "seen" or checked out.

The following sections provide more information on each of these steps.

File Types and Element Types

Each element is an *instance* of an element type (just as each version label is an instance of a label type, each attribute is an instance of an attribute type, and so on). You can specify an element type with the `–eltype` option. (The `lstype -eltype` command lists a VOB's element types.) The element type must already exist in the VOB where you are creating the new element. A `mkelem -eltype directory` command is equivalent to a *mkdir* command.

If you do not specify an element type on the command line, *mkelem* determines one automatically by using *magic* files to perform *file-typing*. This involves a pattern-match on the new element's name (and an examination of the existing view-private file with that name, if one exists — see "Converting View-Private Files to Elements" below). If file-typing fails, an error occurs and no element is created:

```
cleartool: Error: Can't pick element type from rules in ...
```

For more on file-typing, see the *cc.magic* manual page.

Access Mode

Standard UNIX procedure is to create files with mode 666, and directories with mode 777, as modified by your umask. But a file element's "write" access settings are essentially irrelevant — modifications to elements are controlled by ClearCase-level permissions, as described in the *ct_permissions* manual page. When your view selects a checked-in version of a file element, all of its "write" permissions are turned off, corresponding to the fact that the element is read-only. When you checkout an element, "write" permissions are added to the view-private copy. (See the *checkout* manual page for details.)

In view of the fundamental read-only status of file elements, the mode to which your umask is applied is 444 (not 666) for a file element. When you convert a view-private file to an element (see below), its "read" and "execute" rights become those of the new element.

Converting View-Private Files to Elements

You can use the *mkelem* command to convert a view-private file to a file element with the same name. There are several cases:

1. By default, *mkelem* creates an empty version 0 of the new element, then checks out the new element to your view. The view-private file becomes the checked-out version of the new element.
2. If you use the `-ci` option ("checkin"), *mkelem* does all of the above, then proceeds to checkin version 1 of the new element. Thus, version 1 has the contents of the view-private file.
3. If you use the `-nco` option ("no checkout"), *mkelem* just creates an empty version 0 of the new element.

During the element-creation process, *mkelem* renames the view-private file to prevent a name collision that would affect other ClearCase software (for example, triggers on the `mkelem` operation). If this renaming fails, a warning message appears. There are two renaming procedures:

- In cases 1 and 2 above, where a checkout is performed on the new element, *mkelem* temporarily renames the view-private file, using a ***.mkelem*** (or possibly, ***.mkelem***.*n*) suffix. After the new element has been created and checked out, *mkelem* restores the original name. This produces the intended effect: the data formerly in a view-private file is now accessible through an element with the same name.
- In case 3 above, where no checkout is performed on the new element, *mkelem* alerts you that the view-private file has been renamed, using a ***.keep*** (or possibly, ***.keep***.*n*) suffix. Note that the view-private file has not really been converted to an element — it has been moved out of the way to allow creation of an element in its place.

NOTE: If *mkelem* is interrupted, it tries to "clean up". It is possible that your view-private file will be left under the ***.mkelem*** file name.

Auto-Make-Branch During Element Creation

If your config spec has a "/main/LATEST" rule with a `-mkbranch` clause, *mkelem* checks out a subbranch instead of the *main* branch. For example, suppose your view has this config spec: bugfix branch when checked-out:

```
element * CHECKEDOUT
element * .../gopher_port/LATEST
element * V1.0.1 -mkbranch gopher_port
element * /main/LATEST -mkbranch gopher_port
```

In this case, a newly-created element will automatically get a *gopher_port* branch; and this branch will be checked out instead of *main*:

```
Created directory element "release".
% cleartool mkelem –c "new element for Gopher porting work" base.h
Created element "base.h" (type "text_file").
Created branch "gopher_port" from "base.h" version "/main/0".
Checked out "base.h" from version "/main/gopher_port/0".
```

The *auto-make-branch* facility is not invoked if you use the `–nco` option to suppress checkout of the new element. For more on this facility, see the *checkout* and *config_spec* manual pages.

Referencing Element Objects and Their Versions

You sometimes need to distinguish an element itself from the particular version of the element that is selected by your view. In general:

- Appending the extended naming symbol (by default, `@@`) to an element's name references the element itself.
- A simple name (no extended naming symbol) is a reference to the version selected by the view.

For example, *msg.c@@* references an element, whereas *msg.c* references a version of that element. In many contexts (for example, *checkin* and *lsvtree*), ClearCase allows you to ignore the distinction. But there are ambiguous contexts in which you need to be careful. For example, you can attach attributes and hyperlinks either to version objects or to element objects. Thus, these two commands are different:

```
% cleartool mkattr BugNum 403 msg.c        (attaches attribute to version)
% cleartool mkattr BugNum 403 msg.c@@      (attaches attribute to element)
```

NOTE: Do not create elements whose names end with the extended naming symbol. ClearCase cannot handle such elements.

Storage Pools

Physical storage for an element's versions (*data containers*) will be allocated in the storage pools that *mkelem* assigns. You can change pool assignments with the *chpool* command.

Group Membership Restriction

Each VOB has a *group list*. You can create an element in a VOB only if your *principal group* — the first (or only) group listed when you enter an *id*(1) command — is on this list. See the *protectvob* manual page for more on this topic.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, pool (non-directory elements only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Checkout of the New Element. *Default: mkelem* performs a *checkout* on the new element. If a view-private file already exists at that pathname, it becomes the checked-out version of the element. Otherwise, an empty view-private file is created, and becomes the checked-out version.

–nco Suppresses automatic checkout; *mkelem* creates the new element, along with the *main* branch and version */main/0*, but does not check it out. If *element-pname* exists, it is moved aside to a *.keep* file, as explained above.

–ci Creates the new element and version */main/0*, performs a *checkout*, and checks in a new version containing the data in view-private file or derived object *element-pname*, which must exist. You cannot use this option when creating a directory element.

Specifying the Element Type. *Default: mkelem* performs file-typing to select an element type. If file-typing fails, an error occurs. See the *cc.magic* manual page for details on file-typing.

–elt·ype *element-type-name*
Specifies the type of element to be created. The element type must be a ClearCase predefined type, or a user–defined type created with the *mkeltype* command. Specifying `–eltype directory` is equivalent to using the *mkdir* command.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Element(s). *Default:* None.

element-pname …
The pathnames of one or more elements to be created. If you also specify the **–ci** option, each *element-pname* must name an existing view-private object. You cannot create a directory element with the same name as an existing view-private file or directory.

EXAMPLES

- Create a file element named *convolution.c* of type *text_file* (ASCII file type), and checkout the initial version (version 0).

```
% cleartool mkelem –nc –eltype compressed_text_file convolution.c
Created element "convolution.c" (type "compressed_text_file").
Checked out "convolution.c" from version "/main/0".
```

- Create three file elements, *cm_add.c*, *cm_fill.c*, and *msg.c*, allowing the ClearCase magic file to determine the element type(s). Do not checkout the initial versions.

```
% cleartool mkelem –nc –nco cm_add.c cm_fill.c msg.c
Created element "cm_add.c" (type "text_file").
Created element "cm_fill.c" (type "text_file").
Created element "msg.c" (type "text_file").
```

- Convert a view-private file named *test_cmd.c*, to an element, and checkin the initial version.

```
% cleartool mkelem –nc –ci test_cmd.c
Created element "test_cmd.c" (type "text_file").
Checked in "test_cmd.c" version "/main/1".
```

- Create two directory elements and checkout the initial version of each.

```
% cleartool mkelem –nc –eltype directory libs include
Created element "libs" (type "directory").
Checked out "libs" from version "/main/0".
Created element "include" (type "directory").
Checked out "include" from version "/main/0".
```

- Create an element type named *archive* for library archive files, with the user-defined *bin_file* as its supertype. Then, change to the *libs* directory, check it out, and create two elements of type *archive* without checking them out.

```
% cleartool mkeltype –nc –supertype bin_file archive
Created element type "archive".

% cd libs

% cleartool co –nc .
Checked out "." from version "/main/1".

% cleartool mkelem –nc –nco –eltype archive libntx.a libpvt.a
Created element "libntx.a" (type "archive").
Created element "libpvt.a" (type "archive").
```

SEE ALSO

cleartool subcommands: checkout, chpool, lstype, lshistory, mkdir, mkeltype, mkpool, protect, protectvob cc.magic, config_spec, ct_permissions, profile_ccase

NAME mkeltype – create an element type object

SYNOPSIS

mkeltype [**–rep·lace**] **–sup·ertype** *elem-type-name* [**–man·ager** *mgr-name*]
[**–pti·me**] [**–att·ype** *attr-type*[,...]] [**–vob** *pname-in-vob*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *type-name* ...

DESCRIPTION

Creates one or more user-defined *element types* for future use within a VOB. User-defined element types are variants of the ClearCase predefined types (see complete list below). After creating an element type, you can create elements of that type using *mkelem*, or change its type using *chtype*.

Element Supertypes

When you create a new element type, you must specify an existing element type as its *supertype*. The new element type *inherits* the *type manager* of the supertype, unless you use the `–manager` option. The type manager performs such tasks as storing/retrieving the contents of the element's versions (See the *type_manager* manual page.)

For example, you might create an element type *c_source*, with *text_file* as the supertype; *c_source* inherits the type manager associated with the *text_file* supertype — the *text_file_delta* manager.

The `lstype -eltype -long` command lists both the supertype of an element type and its type manager.

Text Files, Cleartext, and a View´s Text Mode

This section applies to the element types *text_file* and *compressed_text_file*, and to all user-defined element types derived from them through the supertype mechanism.

All versions of a text file element are stored together in a single, structured data container file. When a user program accesses a particular version, the type manager:

1. *extracts* the text lines of that particular version from the data container
2. stores the extracted lines in a *cleartext file*, within the cleartext storage pool directory associated with the element
3. arranges for the program to access the cleartext file (not the structured data container)

On subsequent accesses to the same version, steps 1 and 2 can be skipped — the program accesses the existing cleartext file, which is "cached" in the cleartext storage pool.

Operating systems vary in their use of text-file line terminators. To avoid confusion, each ClearCase view has a *text mode*, which determines the line terminator for text files in that view. (See the *mkview* manual page.) After the type manager constructs a cleartext file for a version, its line terminators may be adjusted before the version is presented to the calling program. Adjustment of line terminators can also occur when the *checkout* command copies a version of a text file element, creating a view-private file (the "checked-out version").

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type creator (with –replace only), VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type (with –replace only). See the "Permissions Checking" section of the *cleartool* manual

page.

OPTIONS AND ARGUMENTS

Handling of Name Collisions. *Default:* An error occurs if an element type named *type-name* already exists in the VOB.

–rep·lace Replaces the existing definition of *type-name* with a new one. You must specify *all* options, even those that you wish to preserve from the old definition. You cannot:

- change the type manager (`–manager` or `–supertype` option) if there are existing elements of type *type-name*
- change the definition of a predefined element type (such as *file* or *text_file*)

Supertype / Type Manager Inheritance. *Default:* None — you must specify a supertype. The new element type inherits the type manager of this supertype, unless you use the `–manager` option.

–sup·ertype *elem-type-name*
The name of an existing element type, predefined or user-defined. ClearCase's predefined element types are:

file	Versions can contain any kind of data (ASCII, binary). Uses the *whole_copy* type manager.
compressed_file	Versions can contain any kind of data (ASCII, binary). Uses the *z_whole_copy* type manager.
text_file	All versions must contain ASCII text. Null bytes are not permitted (a byte of all zeros); no line can contain more than 8000 characters. Uses the *text_file_delta* type manager.
compressed_text_file	All versions must contain ASCII text. Uses the *z_text_file_delta* type manager.
directory	Versions of a directory element *catalog* (list the names of) elements and VOB symbolic links. Uses the *directory* type manager, which compares and merges versions of directory elements.

The *lstype* command lists a VOB's existing element types. The listing includes *file_system_object*, but you can only specify `-supertype file_system_object`, if you also specify a type manager with `–manager`.

–man·ager *mgr-name*
Specifies the type manager for the new element type(s), overriding inheritance from the supertype. ClearCase's type managers are:

text_file_delta	Stores all versions in a single structured data container file (similar to an SCCS `s.` file or an RCS `,v` file). Uses incremental file differences to reconstruct individual versions "on the fly". For ASCII files only.

z_text_file_delta	Stores all versions in a single structured data container file, in compressed format. For ASCII files only.
whole_copy	Stores a whole copy of each version in a separate data container file.
z_whole_copy	Stores each version of an element in a separate, compressed data container file. Note that compressed files generally take more time to checkin (since they must be compressed), and reconstruct when first accessed (first cleartext fetch). See *compress*(1) for details on the *z_text_file_delta* and *z_whole_copy* compression algorithm.
directory	Not involved in storing/retrieving directory versions, which reside in the VOB database, not in a source storage pool. This type manager compares and merges versions of the same directory element.

Controlling Version-Creation Time. *Default:* For all elements of the newly created type: whenever a new version is checked in, the version's time-modified stamp is set to the checkin time.

–pti·me For all elements of the newly created type: preserves the time-modified stamp of the checked-out version during *checkin*. In effect, this establishes `checkin -ptime` as the default for elements of this type.

Suggested Attributes. (advisory only, not restrictive) *Default:* The new element type has no list of suggested attributes.

–att·ype *attr-type*[,...]
A comma-separated list (no white space) of existing attribute types. Use this option to inform users of suggested attributes for use with elements of the newly created type. (Users can view the list with *describe* or `lstype -eltype -long`.)

VOB Specification. *Default:* The element type is created in the VOB that contains the current working directory.

–vob *pname-in-vob*
Specifies the VOB in which to create the element type(s). *pname-in-vob* can be the pathname of any object within the VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Element Types. *Default:* None.

type-name …

One or more names for the element types to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

EXAMPLES

- Create an element type named *c_source* using the predefined *text_file* element type as the supertype.

```
% cleartool mkeltype –supertype text_file –nc c_source
Created element type "c_source".
```

- Create an element type for storing binary data named *bin_file*, using the predefined *file* element type as the supertype.

```
% cleartool mkeltype –supertype file –nc bin_file
Created element type "bin_file".
```

- Create an element type based on the user-defined element type *bin_file* (from previous example) for storing executable files. Include an attribute list.

```
% cleartool mkeltype –supertype bin_file –attype Confidence_Level,QAed –nc exe_file
Created element type "exe_file".
```

- Create a "directory of include files" element type, using the predefined *directory* element type as the supertype. Provide a comment on the command line.

```
% cleartool mkeltype –supertype directory –c "directory type for include files" incl_dir
Created element type "incl_dir".
```

- Change the *checkin* default for an existing element type so that it preserves the file modification time. Provide a comment on the command line.

```
% cleartool mkeltype –replace –supertype bin_file –ptime  \
     –c "change archive mod time default" archive
Replaced definition of element type "archive".
```

SEE ALSO

cleartool subcommands: checkin, chtype, describe, lstype, mkattr, mkattype, mkelem, mkview, rmtype, rntype
profile_ccase, type_manager

NAME mkhlink – attach a hyperlink to an object

SYNOPSIS

mkhlink [**–uni·dir**] [**–tte·xt** *to-text*] [**–fte·xt** *from-text*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *hlink-type-name*
from-obj-pname [*to-obj-pname*]

DESCRIPTION

Creates a *hyperlink* between two objects, each of which may be an element, branch, version, or VOB symbolic link.

Logically, a hyperlink is an "arrow" attached to one or two VOB-database objects:

- A *bidirectional* hyperlink connects two objects, in the same VOB or in different VOBs, with optional text annotations at either end. It can be navigated in either direction: "from-object → to-object" or "to-object → from-object".
- A *unidirectional* hyperlink connects two objects in different VOBs, with optional text annotations at either end. It can be navigated only in the "from-object → to-object" direction.
- A *text-only* hyperlink associates one object with a user-defined text string (for example, an element that implements a particular algorithm with the name of a document that describes it).
- A *null-ended* hyperlink has only a from-object. Use this kind of hyperlink to suppress *hyperlink inheritance* (see below).

Contrast with Other Kinds of Meta-Data

Like other kinds of meta-data annotations — version labels, attributes, and triggers — a hyperlink is an instance of a type object: the *mkhlink* command creates an instance of an existing *hyperlink type* object. But hyperlinks differ from other kinds of meta-data annotations:

- The hyperlink created by *mkhlink* is also an object in itself. Each hyperlink object has a unique ID (see next section), and can itself be annotated with attributes. By contrast, a *mklabel*, *mkattr*, or *mktrigger* command does not create a new object — it simply annotates an existing object.
- You can attach several hyperlinks of the same type to one object, but only one instance of a particular label, attribute, or trigger type. (For example, you can attach two different *DesignFor* hyperlinks to the same object, but not two different *ECOnum* attributes.)

Hyperlink-IDs

Each new hyperlink object gets a unique identifier, its *hyperlink-ID*. You can specify any hyperlink by suffixing its hyperlink-ID to the name of the hyperlink type. For example:

```
% cleartool describe DesignFor@52179@/vobs/doctn
```

In this example, "DesignFor" is the name of a hyperlink type, and "@52179@/vobs/doctn" is the hyperlink-ID. Note that the hyperlink-ID includes a pathname — the VOB-tag of the VOB in which the hyperlink is created. When specifying a hyperlink, you can use any pathname within the VOB in place of the VOB-tag pathname:

```
% cd /vobs
% cleartool describe DesignFor@52179@doctn
```

You can omit the pathname altogether if the current working directory is in that VOB:

```
% cd /vobs/doctn
% cleartool describe DesignFor@52179
```

A hyperlink-ID is unique in that it is guaranteed to differ from the hyperlink-ID of all other hyperlinks. But it can change over time — when a VOB's database is processed with *reformatvob*, all hyperlink-IDs get new numeric suffixes:

```
before 'reformatvob':      @52179@/vobs/doctn
after 'reformatvob':       @8883@/vobs/doctn
```

Similarly, the VOB-tag part of a hyperlink-ID can change over time, and can even vary from host to host.

Hyperlink Inheritance

By default, a version implicitly *inherits* a hyperlink attached to any of its ancestor versions, on the same branch or on a parent branch. Inherited hyperlinks are listed by the *describe* command only if you use the `-ihlink` option.

A hyperlink stops being passed down to its descendents if it is superseded by another hyperlink of the same type, explicitly attached to some descendent version. You can use a *null-ended* hyperlink ("from" object, but no "to" object) as the superseding hyperlink to effectively cancel hyperlink inheritance.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Unidirectional/Bidirectional. *Default:* Creates a *bidirectional* hyperlink. If the objects being linked are in different VOBs, then a notation is made in the VOB database of the "to" object, making it possible to see the hyperlink from either VOB.

-uni·dir Creates a *unidirectional* hyperlink; no notation is made in the VOB database of the "to" object (if that object is in a different VOB).

NOTE: In all cases, a single hyperlink object is created, in the VOB of the "from" object.

Text Annotations. *Default:* The hyperlink has no text annotations.

-tte·xt *to-text*
Text associated with "to" end of a hyperlink. If you also specify *to-obj-pname*, the text is associated with that object. If you do not specify *to-obj-pname*, *cleartool* creates a *text-only* hyperlink, originating from *from-obj-pname*. If you omit both `-ttext` and *to-obj-pname*, *cleartool* creates a *null-ended* hyperlink.

-fte·xt *from-text*
Text associated with "from" end of a hyperlink.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `-nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

-c *comment* , **-cq** , **-cqe** , **-nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Hyperlink Type. *Default:* None.

hlink-type-name
An existing hyperlink type. If the objects being linked are in different VOBs, *hlink-type-name* must exist in both VOBs (unless you use the `-unidir` option).

Objects to be Hyperlinked. *Default:* None — you must specify at least one object.

from-obj-pname
Pathname of the from-object:

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A version-extended pathname specifies an element, branch, or version, independent of view.
- The pathname of a VOB symbolic link.

Examples:

`foo.c`	*(version of 'foo.c' selected by current view)*
`/view/gamma/usr/project/src/foo.c`	*(version of 'foo.c' selected by another view)*
`foo.c@@/main/5`	*(version 5 on main branch of 'foo.c')*
`foo.c@@/REL3`	*(version of 'foo.c' with version label 'REL3')*
`foo.c@@`	*(the element 'foo.c')*
`foo.c@@/main`	*(the main branch of element 'foo.c')*

to-obj-pname
(optional) Pathname of the to-object, specified in the same way as the from-object. Omitting this argument creates a *text-only* hyperlink (if you use `-ttext`) or a *null-ended* hyperlink (if you don't).

NOTE: An error occurs if you try to make a unidirectional hyperlink whose *to-obj-pname* is a checked-out version in another VOB.

EXAMPLES

- Create a hyperlink type; then create a unidirectional, element-to-element hyperlink between an executable and its GUI counterpart in another VOB.

```
% cleartool mkhltype -nc gui_tool
Created hyperlink type "gui_tool".

% cleartool mkhlink -unidir gui_tool monet@@ /vobs/gui/bin/xmonet@@
Created hyperlink "gui_tool@1239@/usr/hw".
```

- Create a hyperlink of type *design_spec* connecting the versions of a source file and design document labeled *REL2*.

```
% cleartool mkhlink design_spec util.c@@/REL2 \
      /usr/hw/doc/util.doc@@/REL2
Created hyperlink "design_spec@685@/usr/hw".
```

- Create three hyperlinks of the same type from the same version of a design document; each hyperlink points to a different source file element.

```
% cleartool mkhlink design_for sortmerge.doc ../src/sort.c
Created hyperlink "design_for@4249@/vobs/proj".

% cleartool mkhlink design_for sortmerge.doc ../src/merge.c
Created hyperlink "design_for@4254@/vobs/proj".

% cleartool mkhlink design_for sortmerge.doc ../src/sortmerge.h
Created hyperlink "design_for@4261@/vobs/proj".
```

- Create an element-to-element hyperlink between a source file and a script that tests it. Specify both "from text" and "to text" for further annotation.

```
% cleartool mkhlink –ttext "regression A" –ftext "edge effects" \
      tested_by cm_add.c@@ edge.sh@@
Created hyperlink "tested_by@714@/usr/hw".
```

- Create a hyperlink of type *fixes* between the version of *util.c* in your view and the element *bug.report.21*. Use "to text" to indicate the bug number ("fixes bug 21").

```
% cleartool mkhlink –ttext "fixes bug 21" fixes  \
      util.c /usr/hw/bugs/bug.report.21@@
Created hyperlink "fixes@746@/usr/hw".
```

- Create a *text only* hyperlink of type *design_spec* to associate the algorithm *convolution.c* with the third party document describing that algorithm. Make the hyperlink between the element *convolution.c* and the "to text" that describes it.

```
% cleartool mkhlink –ttext "Wilson: Digital Filtering, p42–50" \
      design_spec convolution.c@@
Created hyperlink "design_spec@753@/usr/hw".
```

SEE ALSO

cleartool subcommands: describe, lstype, mkhltype, rmhlink, rmtype, rntype
xclearcase, profile_ccase

NAME mkhltype – create a hyperlink type object

SYNOPSIS

mkhltype [**–rep·lace**] [**–att·ype** *attr-type*[,...]] [**–vob** *pname-in-vob*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–oma·ster**] *type-name* ...

DESCRIPTION

Creates one or more *hyperlink types* for future use within a VOB. After creating a hyperlink type, you can connect pairs of objects with hyperlinks of that type, using *mkhlink*.

Conceptually, a *hyperlink* is an "arrow" from one *VOB-database object* (version, branch, element, or VOB symbolic link) to another. To enable objects in two different VOBs to be connected, a hyperlink with the same name must be created in both VOBs.

For example, you might create a hyperlink type named *design_spec*, for use in linking source code files to the associated design documents. Later, you might use *mkhlink* to create a hyperlink of this type between *my_prog.c* and *my_prog.dsn*.

Predefined Hyperlink Type

Each new VOB is created with a hyperlink type named *Merge*. When you perform a merge of two or more versions of an element with the *merge* command, your work is record with one or more *merge arrows*. Each merge arrow is actually a hyperlink of type *Merge*, connecting one of the contributors to the target version.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: (with –replace only): type creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, hyperlink type (with –replace only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of Name Collisions. *Default:* An error occurs if a hyperlink type named *type-name* already exists in the VOB.

–rep·lace Replaces the existing definition of *type-name* with a new one. You must specify *all* options, even those that you wish to preserve from the old definition. Additional restrictions:

- You cannot replace the predefined hyperlink type *Merge*.
- When replacing a hyperlink type that was created with the `–omaster` option, you must use `–omaster` again; that is, you cannot convert a hyperlink type from shared to unshared.

Suggested Attributes. (advisory only, not restrictive) *Default:* The new hyperlink type has no list of suggested attributes.

–att·ype *attr-type*[,...]
A comma-separated list (no white space) of existing attribute types. Use this option to inform users of suggested attributes for use with hyperlinks of the newly created type. (Users can view the list with *describe* or `lstype -hltype -long`.)

In this release, *mkattr* attaches attributes only to file system objects, not to hyperlinks.

VOB Specification. *Default:* The element type is created in the VOB that contains the current working directory.

–vob *pname-in-vob*
Specifies the VOB in which to create the hyperlink type(s). *pname-in-vob* can be the pathname of any object within the VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Hyperlink Types. *Default:* None.

type-name …
One or more names for the hyperlink types to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

Mastership of the Hyperlink Type. Attempts to attach hyperlinks of this type will succeed only in the *VOB replica* which is the current *master* of the hyperlink type. The VOB replica in which the new hyperlink type is created becomes its initial master.

–oma·ster Hyperlinks of this type can be created in any VOB replica.

EXAMPLES

- Create a hyperlink type named *tested_by*.

```
% cleartool mkhltype –nc tested_by
Created hyperlink type "tested_by".
```

- Create a hyperlink type named *design_spec*, and provide a comment on the command line.

```
% cleartool mkhltype –c "source to design document" design_spec
Created hyperlink type "design_spec".
```

- Create a hyperlink type named *test_script*, providing a suggested-attribute list.

```
% cleartool mkhltype –nc –attype run_overnight,error_rate test_script
Created hyperlink type "test_script".
```

SEE ALSO

cleartool subcommands: describe, lstype, mkattr, mkattype, mkhlink, rmtype, rntype
events_ccase, profile_ccase

NAME mklabel – attach version labels to versions of elements

SYNOPSIS

- Attach label to specified versions:

  mklabel [**–rep·lace**] [**–r·ecurse**] [**–ver·sion** *version-selector*]
  [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *label-type-name pname …*

- Attach label to versions listed in configuration record:

  mklabel [**–rep·lace**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  –con·fig *do-pname* [**–sel·ect** *do-leaf-pattern*] [**–ci**]
  [**–typ·e** { **f** | **d** } …] [**–nam·e** *tail-pattern*] *label-type-name*

DESCRIPTION

Prerequisite: A 'label type' object, created with 'mklbtype', must already exist in the VOB(s) containing the versions to be labeled.

Attaches a *version label* to one or more versions. You can specify the versions themselves on the command line, or you can specify a particular derived object. In the latter case, *mklabel* labels some or all the versions that were used to build that derived object.

Referencing Labeled Versions

Labeling a version of an element can affect the way the element appears in views. It also provides a new way to access the version with a version-extended pathname.

Version Selection by Views. A typical ClearCase config spec rule uses version labels to select versions:

```
element * BASELEVEL_1
```

If you attach version label *BASELEVEL_1* to a version of element *foo.c*, then any view configured with this rule will select the labeled version (unless some rule earlier in the config spec matches another version of *foo.c*).

Version Labels in Version-Extended Pathnames. Labeling a version effectively adds a new *hard link* to the version in ClearCase's extended namespace. If you attach version label *R4.1A* to version */main/rls4/12* of element *bar.c*, then these pathnames are equivalent:

```
bar.c@@/main/rls4/12
bar.c@@/main/rls4/R4.1A
```

In addition, a third pathname is *usually* equivalent:

```
bar.c@@/R4.1A
```

This version-extended pathname is valid if it is unambiguous — that is, if no other version of *bar.c* is currently labeled *R4.1A*. (This is usually the case since, by default, label types are restricted to being used once-per-element. See the description of the `–pbranch` option in the *mklbtype* manual page.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, label type. See the "Permissions Checking"

section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Moving a Version Label. *Default:* An error occurs if a version label of this type is already attached to some other version of the same element.

–rep·lace Removes an existing label of the same type from another version of the element:

- from another version on the same branch, if *label–type–name* was created with `mklbtype -pbranch`
- from another version anywhere in the element's version tree, *label-type-name* was not created with `mklbtype -pbranch`

No error occurs if there is no such label to remove.

Specifying the Label Type. *Default:* None.

label-type-name
A label type, previously created with *mklbtype*. If the versions to be labeled reside in different VOBs, the label type must exist in each VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Directly Specifying the Versions to be Labeled. The options and arguments in this section specify elements and their versions directly on the command line. Do not use these options and arguments when using a derived object to provide a list of versions.

pname … (required) One or more pathnames, indicating versions to be labeled:

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A version-extended pathname specifies a version, independent of view.

Use `–version` to override these interpretations of *pname*.

–ver·sion *version-selector*
For each *pname*, attaches the label to the version specified by *version-selector*. This option overrides both version–selection by the view and version–extended naming. See the *version_selector* manual page for syntax details.

–r·ecurse Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are *not* traversed during the recursive descent into the subtree.

NOTE: *mklabel* differs from some other commands in its default handling of directory element *pname* arguments: it labels the directory element itself; it does not label the elements cataloged in the directory.

Using a Derived Object to Specify the Versions to be Labeled. The options and arguments in this section specify versions by selecting them from the configuration record(s) associated with a particular derived object. Do not use these options when specifying elements and versions directly on the command line.

–con·fig *do-pname*
(required) Specifies one derived object. A standard pathname or view-extended pathname specifies the DO that currently appears in a view. To specify a DO independent of view, use an extended name that includes a *DO-ID* (for example, `hello.o@@24-Mar.11:32.412`) or a version-extended pathname to a DO version.

With the exception of checked-out versions, *mklabel* labels all the versions that would be included in a `catcr -flat` listing of that derived object. Note that this includes any DO created by the build and subsequently checked in as a DO version.

If the DO's configuration includes multiple versions of the same element, only the most recent version is labeled.

Use the following options to modify the list of versions to be labeled.

–sel·ect *do-leaf-pattern*
–ci
–nam·e *tail-pattern*
–typ·e { **f** | **d** } ...
Modify the set of versions to be labeled in the same way that these options modify a *catcr* listing. See the *catcr* manual page for details, along with the "Examples" section below.

EXAMPLES

- Create a label type named *REL6*. Then, attach that label to all versions at or below the current working directory.

```
% cleartool mklbtype –nc REL6
Created label type "REL6".

% cleartool mklabel –recurse REL6 .
Created label "REL6" on "." version "/main/4".
Created label "REL6" on "./bin" version "/main/1".
Created label "REL6" on "./include" version "/main/1".
Created label "REL6" on "./libs" version "/main/2".
Created label "REL6" on "./lost+found" version "/main/0".
Created label "REL6" on "./release" version "/main/1".
Created label "REL6" on "./src" version "/main/6".
Created label "REL6" on "./libs/libntx.a" version "/main/1".
Created label "REL6" on "./libs/libpvt.a" version "/main/3".
Created label "REL6" on "./src/Makefile" version "/main/2".
Created label "REL6" on "./src/cm_add.c" version "/main/1".
Created label "REL6" on "./src/cm_fill.c" version "/main/2".
Created label "REL6" on "./src/convolution.c" version "/main/4".
Created label "REL6" on "./src/edge.sh" version "/main/1".
  ⋮
```

- Attach label *REL1* to the version of *msg.c* in the view.

```
% cleartool mklabel REL1 msg.c
Created label "REL1" on "msg.c" version "/main/1".
```

- Attach label *REL2* to version 3 on the *rel2_bugfix* branch of file *util.c*.

```
% cleartool mklabel –version /main/rel2_bugfix/3 REL2 util.c
Created label "REL2" on "util.c" version "/main/rel2_bugfix/3".
```

- Move label *REL2* to a different version of element *hello.c*, using a version-extended pathname to indicate that version.

```
% cleartool mklabel –replace REL2 hello.c@@/main/4
Moved label "REL2" on "hello.c" from version "/main/3" to "/main/4".
```

- Attach label *REL3* to each version that was used to build derived object *hello.o*. Note that both directories and files are labeled.

```
% cleartool mklabel –config hello.o REL3
Created label "REL3" on "/usr/hw/" version "/main/1".
Created label "REL3" on "/usr/hw/src" version "/main/2".
Created label "REL3" on "/usr/hw/src/hello.c" version "/main/3".
Created label "REL3" on "/usr/hw/src/hello.h" version "/main/1".
```

- Attach label *REL5* to each C-language source file version that was used to build derived object *hello*.

```
% cleartool mklabel –config hello –name '*.c' REL5
Created label "REL5" on "/usr/hw/src/hello.c" version "/main/3".
Created label "REL5" on "/usr/hw/src/util.c" version "/main/1".
```

- Attach label *REL5* to all versions in the VOB mounted at */usr/hw* that were used to build derived object *hello*. Use interactive mode to enable use of the ClearCase "..." wildcard.

```
% cleartool
cleartool> mklabel –config hello –name '/usr/hw/...' REL5
Created label "REL5" on "/usr/hw/" version "/main/1".
Created label "REL5" on "/usr/hw/src" version "/main/2".
Created label "REL5" on "/usr/hw/src/hello.c" version "/main/3".
Created label "REL5" on "/usr/hw/src/hello.h" version "/main/1".
Created label "REL5" on "/usr/hw/src/util.c" version "/main/1".
```

SEE ALSO

cleartool subcommands: catcr, lsdo, mklbtype, rmlabel, rmtype, rntype
profile_ccase, version_selector

NAME mklbtype – create a label type object

SYNOPSIS

mklbtype [**–rep·lace**] [**–pbr·anch**] [**–vob** *pname-in-vob*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–oma·ster**] *type-name* …

DESCRIPTION

Creates one or more *label types* with the specified names for future use within a VOB. After creating a label type in a VOB, you can attach labels of that type to versions of that VOB's elements, using *mklabel*.

Instance Restrictions

ClearCase can allow the same version label to be attached to multiple versions of the same element. (The versions must all be on different branches — if there were two versions labeled *JOHN_TMP* on branch */main/bugfix*, then the version-extended pathname `foo.c@@/main/bugfix/JOHN_TMP` would be ambiguous.) But there are drawbacks to using the same version label several times in the same element:

- It is potentially confusing.
- In a version-extended pathname, you must always include a full branch pathname along with the version label (for example, `foo.c@@/main/rs6000_port/JOHN_TMP`).

By default, a new label type carries the restriction that it can be used on only one version in an element's entire version tree. This allows you to omit the branch pathname portion of a version-extended pathname (for example, `foo.c@@/JOHN_TMP`). The `–pbranch` option relaxes this restriction, allowing the label type to be used once-per-branch.

Recommended Naming Convention

A VOB cannot contain a branch type and a label type with the same name. For this reason, we strongly recommend that you adopt this convention:

- Make all letters in names of branch types lowercase (`a – z`).
- Make all letters in names of label types uppercase (`A – Z`).

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: (with –replace only): type creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, label type (with –replace only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of Name Collisions. *Default:* An error occurs if a label type named *type-name* already exists in the VOB.

–rep·lace Replaces the existing definition of *type-name* with a new one. You must specify *all* options, even those that you wish to preserve from the old definition. Additional restrictions:

– You cannot replace either of the predefined label types *LATEST* and *CHECKEDOUT*.

- If there are existing labels of this type, you cannot replace a less restrictive definition (–pbranch specified) with a more restrictive definition (the default once-per-element).
- When replacing a label type that was created with the –omaster option, you must use –omaster again; that is, you cannot convert a label type from shared to unshared.

Instance Restrictions. *Default:* A label of the new type can be attached to only one version of a given element.

–pbr·anch Relaxes the default restriction, allowing the label type to be used once-per-branch in a given element's version tree. ClearCase never allows the same version label to be attached to multiple versions on the same branch.

VOB Specification. *Default:* The label type is created in the VOB that contains the current working directory.

–vob *pname-in-vob*
Specifies the VOB in which to create the label type(s). *pname-in-vob* can be the pathname of any object within the VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –cqe). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Label Types. *Default:* None.

type-name …
One or more names for the label types to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

See "Recommended Naming Convention" above.

Mastership of the Label Type. Attempts to attach or remove labels of this type will succeed only in the *VOB replica* which is the current *master* of the label type. The VOB replica in which the new label type is created becomes its initial master.

–oma·ster Mastership of this label type will be ignored in *mklabel* and *rmlabel* commands; instead, mastership of the version's branch will determine whether the command succeeds.

EXAMPLES

- Create a label type that can be used only once-per-element. Provide a comment on the command line.

```
% cleartool mklbtype –c "Version label for V2.7.1 sources" V2.7.1
Created label type "V2.7.1".
```

- Create a label type that can be used once-per-branch in any element's version tree.

```
% cleartool mklbtype –nc –pbranch REL3
Created label type "REL3".
```

- Change the usage restriction of an existing label type so that it can be used once-per-branch. Provide a comment on the command line. (This change does not affect existing labels of this type.)

```
% cleartool mklbtype –replace –pbranch –c "allow use on multiple branches" V2.7.1
Replaced definition of label type "V2.7.1".
```

SEE ALSO

cleartool subcommands: describe, lstype, mklabel, rmtype, rntype
events_ccase, profile_ccase

NAME mkpool – create a VOB storage pool or modify its scrubbing parameters

SYNOPSIS

- Create source pool:

  mkpool **−sou·rce** [**−ln** *pname*] [**−vob** *pname-in-vob*]
  [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *pool-name …*

- Create derived object pool or cleartext pool:

  mkpool { **−der·ived** | **−cle·artext** } [**−ln** *pname*]
  [**−siz·e** *max-kbytes reclaim-kbytes* [**−age** *hours*] [**−ale·rt** *command*]]
  [**−vob** *pname-in-vob*] [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *pool-name …*

- Update pool parameters:

  mkpool **−upd·ate** [**−siz·e** *max-kbytes reclaim-kbytes*] [**−age** *hours*]
  [**−ale·rt** *command*] [**−vob** *pname-in-vob*]
  [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *pool-name …*

DESCRIPTION

Creates a *source storage pool*, *derived object storage pool*, or *cleartext storage pool*, and initializes the pool's *scrubbing* parameters. You can also use this command to update the scrubbing parameters of an existing storage pool.

Storage pools are directories used as physical storage areas for different kinds of ClearCase data:

- A *source storage pool* stores the *data containers* that contain versions of elements.
- A *derived object storage pool* stores *shared derived objects* — those that are referenced by more than one view.
- A *cleartext storage pool* is a cache of text files. If all of an element's versions are stored in a single data container (in compressed format or *delta* format), accessing a particular version involves some processing overhead — a *type manager* program is invoked to extract the *cleartext* of that version from the data container. As a performance optimization, ClearCase "caches" the extracted version as a file in a cleartext storage pool. The next access to that same version uses the cached copy, saving the cost of extracting the version from the data container again.

Creating a new VOB with the *mkvob* command automatically creates one default pool of each kind: *sdft* (source pool), *ddft* (derived object pool), and *cdft* (cleartext pool).

By default, *mkpool* creates a storage pool as a directory within the VOB storage area. Source pools are always created within subdirectory *s* of the VOB storage directory; derived object pools are created within subdirectory *d*; cleartext pools are created within subdirectory *c*. The `-ln` option allows you to create pool(s) elsewhere, to be accessed at the standard locations through symbolic links.

Pool Allocation and Inheritance

Each file element is assigned to one source pool and one cleartext pool. The source pool provides permanent storage, in one or more data container files, for all of the element's versions. If a single data container stores all the versions, the cleartext pool is used to cache extracted versions of that element, as described above. (If each version is stored in a separate data container, the cleartext pool is not used at all.)

Each directory element is also assigned to one source pool and one cleartext pool. But directory versions themselves are not stored in these pools. (They are stored directly in the VOB database.) Rather, a directory's pool assignments are used solely for *pool inheritance*: each element created within the directory inherits its source and cleartext pool assignments.

Each directory element is also assigned to one derived object pool. All shared derived objects with pathnames in that directory are stored in that pool. A new directory element inherits the derived object pool of its parent, along with the source and cleartext pools.

The pool inheritance scheme begins at the VOB root directory (top–level directory element) created by *mkvob*, which is automatically assigned to the default pools.

You can change any of an element's pool assignments with the *chpool* command.

Scrubbing

Scrubbing is the process of reclaiming space in a derived object pool or cleartext pool. (Source pools are not subject to scrubbing.) This process is performed by the *scrubber* utility which, by default, is run daily by a *crontab*(1) script. *mkpool* initializes or updates these scrubbing parameters:

maximum size	(*max-kbytes*) maximum pool size
reclaim size	(*reclaim-kbytes*) size to which *scrubber* should attempt to reduce the pool
age	(*hours*) threshold to prevent premature scrubbing of recently-referenced objects

The default settings for the scrubbing parameters are: *max-kbytes* = 0, *reclaim-kbytes* = 0, *hours* = 96. See the *scrubber* manual page for details on how these parameters are interpreted. See the *crontab_ccase* manual page for a description of the default automatic scrubbing procedure.

Getting Information on Storage Pools

The *lspool* command lists a VOB's storage pools. If you include the `–long` option, the current settings of the scrubbing parameters are listed, as well. (The `describe -pool` command displays the same information as `lspool -long`.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, root user.
Locks: An error occurs if any of the following objects are locked: VOB, pool (for –update only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Storage Pool / Specifying an Update. *Default:* You must specify the kind of pool, unless you use `-update` and name an existing pool. The following options are mutually exclusive.

-sou·rce Creates a source pool.

-der·ived Creates a derived object pool.

-cle·artext Creates a cleartext pool.

-upd·ate Asserts that the parameters of an existing pool are to be updated. You must also use a `-size` and/or `-age` option.

Specifying New Parameters. *Default:* For a new derived object or cleartext pool: the *maximum size* and *reclaim size* parameters are set to 0, which enables a special scrubbing procedure. (See the *scrubber* manual page.) The *age* parameter is set to 96 (hours). These parameters are meaningless for a source pool.

When updating an existing pool, you must use at least one of `-size` and `-age`.

-siz·e *max-kbytes reclaim-kbytes*
Specifies that the pool will be scrubbed if its size exceeds *max-kbytes* Kb; scrubbing will continue until the pool reaches the goal size of *reclaim-kbytes* Kb.

-age *hours* Prevents scrubbing of derived objects or cleartext files that have been referenced within the specified number of hours.

Special Case: `-age 0` restores the default age setting (96 hours).

Local vs. Remote Storage. *Default:* Creates a storage pool as a subdirectory under the VOB storage directory.

-ln *pname* Creates a storage pool directory at *pname*, and creates *pool-name* in the VOB storage directory as a symbolic link to *pname*. You can create only one pool when using this option.

RESTRICTION: *pname* must be a full pathname, starting with a slash character (`/`). It must also be a *global pathname*, valid on every host from which users will access the VOB. *mkpool* attempts to verify the "globalness" of this pathname, using a simple heuristic. (For example, a pathname that begins with `/net` is likely to be global.) If it suspects that *pname* is not global, *mkpool* proceeds anyway, but displays a warning message:

```
Warning: Linktext for pool does not appear to be a global path.
```

This mechanism is independent of the ClearCase *network storage registry* facility. This, the pathname to a remote storage pool directory must be truly global, not just global within a particular network region.

Scrubber Failure Processing. *Default:* If *scrubber* fails to scrub a pool below its *max-kbytes* level, it logs a warning message in */usr/adm/atria/log/scrubber_log*, but takes no other action.

-ale·rt *command*
Causes *scrubber* to run the specified command (typically, a shell script) whenever it fails to scrub a pool below its *max-kbytes* level.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `-cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

-c *comment* , **-cq** , **-cqe** , **-nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the VOB. *Default:* Creates or updates a pool in the VOB containing the current working directory.

-vob *pname-in-vob*
The VOB whose pool is to be created or updated. *pname-in-vob* can be the pathname of any object within the VOB.

Specifying the Pool. *Default:* None.

pool-name …
One or more names for the storage pools to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

EXAMPLES

- Create a source pool that uses the default pool parameters.

```
% cleartool mkpool -source -c "pool for c source files" c_pool
Created pool "c_pool".
```

- Create a derived object pool with a maximum size of 10000Kb (10Mb) and a reclaim size of 8000Kb (8Mb). Allow the *age* parameter to assume its default value.

```
% cleartool mkpool -derived -nc -size 10000 8000 do1
Created pool "do1".
```

- Update the derived object pool created in the previous example, so that any derived object referenced within the last week (168 hours) will not be scrubbed.

```
% cleartool mkpool -nc -update -age 168 do1
Updated pool "do1".
```

- Create a non-local cleartext storage pool at the globally-accessible location */usr/vobstore/ccase_pools/c2*, to be accessed as pool *cltxt2*.

```
% cleartool mkpool -nc -cleartext -ln /usr/vobstore/ccase_pools/c2 cltxt2
Created pool "cltxt2".
```

This command creates this symbolic link:

vob-storage-dir-pname`/c/cltxt  -> /usr/vobstore/ccase_pools/c2`

- Create a cleartext pool named *my_ctpool* that uses the default pool parameters. Then, change all elements using pool *cdft* (the default cleartext pool) to use *my_ctpool* instead.

```
% cleartool mkpool –cleartext –c "alternate cleartext pool" my_ctpool
Created pool "my_ctpool".

% cleartool find . –all –element 'pool(cdft)' \
      –exec 'cleartool chpool –force my_ctpool $CLEARCASE_PN'
Changed pool for "/usr/hw" to "my_ctpool".
Changed pool for "/usr/hw/bin" to "my_ctpool".
Changed pool for "/usr/hw/bin/hello" to "my_ctpool".
Changed pool for "/usr/hw/bugs" to "my_ctpool".
Changed pool for "/usr/hw/bugs/bug.report.21" to "my_ctpool".
Changed pool for "/usr/hw/doc" to "my_ctpool".
Changed pool for "/usr/hw/doc/util.doc" to "my_ctpool".
Changed pool for "/usr/hw/include" to "my_ctpool".
Changed pool for "/usr/hw/libs" to "my_ctpool".
Changed pool for "/usr/hw/libs/libntx.a" to "my_ctpool".
Changed pool for "/usr/hw/libs/libpvt.a" to "my_ctpool".
  ⋮
```

SEE ALSO

cleartool subcommands: chpool, find, lsdo, lspool, mkelem, mkdir, mkvob
crontab_ccase, profile_ccase, scrubber

NAME mktag – create a view-tag or a public/private VOB-tag

SYNOPSIS

- Create a view-tag:

  mktag **–vie·w** **–tag** *view-tag* [**–tco·mment** *tag-comment*] [**–rep·lace**] [**–nst·art**]
  [**–reg·ion** *network-region*]
  [**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
  view-storage-dir-pname

- Create a VOB-tag:

  mktag **–vob** **–tag** *vob-tag* [**–tco·mment** *tag-comment*] [**–rep·lace**]
  [**–opt·ions** *mount-options*] [**–pub·lic** [**–pas·sword** *tag-registry-password*]]
  [**–reg·ion** *network-region*]
  [**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
  vob-storage-dir-pname

DESCRIPTION

For an existing view or VOB, creates or replaces an entry in the network's *view_tag* or *vob_tag* registry file. A view or VOB gets one tag when it is created with *mkview* or *mkvob*. The principal use of *mktag* is to create additional tags, enabling access from multiple *network regions*. Each network region needs its own tag for a view or VOB. A single region cannot have multiple tags for the same view or VOB. However, a single tag can be assigned to multiple regions with multiple *mktag* commands. See the *registry_ccase* manual page for a discussion of network regions.

Activating the View or VOB

By default, creating a view-tag activates the view on your host, by implicitly performing a *startview* command. This does not occur if your host is not in the tag's assigned network region, or if you use the `–nstart` option.

Creating a VOB-tag does not automatically activate the VOB; use `cleartool mount` for this purpose.

PERMISSIONS AND LOCKS

Only a VOB's owner can create a private VOB-tag. Locks do not apply to this command.

OPTIONS AND ARGUMENTS

Specifying the Kind of Tag. *Default:* None.

–vie·w Creates or updates a view-tag.

–vob Creates or updates a VOB-tag.

Specifying the Tag. *Default:* None.

–tag *view-tag*
A name for the view, in the form of a simple file name.

–tag *vob-tag*
A standard full pathname, which specifies the location at which the VOB will be mounted as a file system of type MVFS.

–tco·mment *tag-comment*
Adds a comment to the tag's entry in the *vob_tag* or *view_tag* registry file. Use the `–long` option on *lsvob* or *lsview* to display the tag comment.

Overwriting an Existing Tag. *Default:* An error occurs if the view or VOB already has a tag in the target network region.

–rep·lace Replaces an existing tag registry entry with a new entry. (No error occurs if the tag does not currently exist.) The `–replace` option is provided to convert private VOBs to public and vice versa, and also to change miscellaneous parameters associated with a tag (tag comment, access paths, mount options, and startview behavior). You *cannot* use `–replace` to change a tag's region, or to change an existing tag's name. To perform either of these operations, you must first delete the existing tag explicitly with *rmtag*.

Starting the View. *Default:* For a view-tag, the view is started on your host, making *view-tag* appear as a directory entry in the ClearCase viewroot directory, */view*. If necessary, a *view_server* process is started on the host where the view storage directory resides.

–nst·art Suppresses starting of the view.

Specifying a Network Region. *Default:* Creates a tag in the local host's network region. (A host's network region is listed in file */usr/adm/atria/rgy/rgy_region.conf*.) See the *registry_ccase* manual page for a discussion of *network regions*.

–reg·ion *network-region*
Creates the tag in the specified *network region*. An error occurs if the region does not already exist. An error occurs if the view or VOB already has a tag in the specified network region.

Specifying Mount Options. *Default:* No mount options are included in the VOB registry entry for a new VOB-tag.

–opt·ions *mount-options*
(VOB-tags only; *root* user only) Specifies mount options to be invoked when the VOB is activated through this VOB-tag. See the *mkvob* manual page for syntax details.

Public vs. Private VOB. *Default:* Creates a private VOB-tag (does not apply to view-tags). An error occurs if you are not the VOB's owner.

–pub·lic Creates a public VOB-tag. See the *mkvob* manual page for a discussion of public and private VOBs.

–pas·sword *tag-registry-password*
Specifies the *VOB–tag password*, which is required to create a public VOB–tag. If a `mktag -vob -public` command line does not include a password, ClearCase prompts for it. The password is checked against the (encrypted) contents of file */usr/adm/atria/rgy/vob_tag.sec*; an error occurs if there is no match. See the *registry_ccase* manual page.

NOTE: The VOB-tags for a given VOB should all be *private*, or all be *public*.

Specifying the Location of the Storage Directory. *Default:* None. You must specify the location of the VOB or view storage directory. In rare cases, you must specify the host name and local/global access path information as well. See the *mkview* and *mkvob* manual pages for details.

view-storage-dir-pname
vob-storage-dir-pname
–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*

EXAMPLES

- For the network region *europe*, assign the new view-tag *view5* to an existing view storage area.

```
% cleartool mktag –view –tag view5 –region europe /net/gw/host3/view_store/view5.vws
```

- For the network region *europe*, register an existing VOB with a public VOB-tag.

```
% cleartool mktag –vob –tag /vobs/us_east1 –region europe –public \
      –password tagPword /net/gw/host2/vob_store/vob1.vbs
```

- Reassign an existing view-tag, *proj2*, to a different view-storage directory.

```
% cleartool mktag –view –tag proj2 –replace /net/host3/view_store/proj2.B.vws
```

- Convert a private VOB to a public VOB, by replacing its private VOB-tag with public one.

```
% mktag –vob –tag /vobs/publicvob –replace –public –pass tagPword /vobs/private.vbs
```

SEE ALSO

cleartool subcommands: lsview, lsvob, mkview, mkvob, pwv, rmtag, setview, startview
filesys_ccase, registry_ccase, view_server

NAME mktrigger – attach a trigger to an element

SYNOPSIS

mktrigger [**–r·ecurse**] [**–nin·herit** | **–nat·tach**] [**–for·ce**]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *trigger-type-name pname …*

DESCRIPTION

Prerequisite: A 'trigger type' object, created with 'mktrtype –element', must already exist in the VOB(s) containing the specified elements.

Attaches a trigger to one or more elements. An attached trigger *fires* (executes the trigger action) when the element or any of its versions is involved in an operation specified in the trigger type definition. For example, if a trigger type is defined to fire on a *checkin* operation, then the attached trigger fires when the specified element is checked in. If a VOB operation causes multiple attached triggers to fire, the order of firing is undefined.

Trigger Inheritance

ClearCase has a *trigger inheritance* scheme, whereby newly-created elements (but *not* existing elements) *inherit* the triggers that are currently associated with their parent directory element. But a simple inherit-all-triggers strategy does not suit the needs of many sites. For example:

- You may want some of a directory's triggers not to propagate to its subtree.
- You may want some triggers to fire only for file elements, not for directory elements.

To enable such flexibility, each directory element has two independent lists of trigger types:

- Its *attached list* specifies triggers that will fire on operations involving the directory element.
- Its *inheritance list* specifies triggers to be inherited by elements created within the directory.

By default, attaching a trigger to a directory element updates both lists:

```
% cleartool mktrigger trig_co proj
Added trigger "trig_co" to inheritance list of "proj".
Added trigger "trig_co" to attached list of "proj".
```

Each file element has only an attached list:

```
% cleartool mktrigger trig_co util.c
Added trigger "trig_co" to attached list of "util.c".
```

You can use the `–ninherit` and `–nattach` options to control exactly which triggers on a directory element will be inherited. (And you can make adjustments using the `–ninherit` and `–nattach` options of the *rmtrigger* command.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, trigger type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Attaching Triggers to an Entire Subdirectory Tree. *Default:* If a *pname* argument names a directory element, the trigger is attached only to the element itself, not to any of the existing elements within it.

–r·ecurse Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are *not* traversed during the recursive descent into the subtree.

Controlling Trigger Inheritance. *Default:* For a directory element, the specified trigger type is placed both on the element's attached list and its inheritance list. (For a file element, the trigger type is placed on its attached list — its only trigger-related list.) The following options apply to directory elements only.

–nin·herit The trigger is placed on the element's attached list only, not on its inheritance list. This option is useful when you wish to monitor operations on a directory, but not operations on the files within the directory.

–nat·tach The trigger is placed on the element's inheritance list only, not on its attached list. This option is useful when you wish to monitor operations on the files within a directory, but not operations on the directory itself.

Observing Element Type Restrictions. *Default:* If *trigger-type-name* is defined with a restriction to one or more element types, *mktrigger* refuses to process an element of another type.

–for·ce Attaches a trigger to an element whose type does not match the definition of the trigger type. Such a trigger will not fire unless you changes the element's type (*chtype*) or you redefine the trigger type (`mktrtype -replace`).

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Trigger Type. *Default:* None.

trigger-type-name
The name of an existing *element* trigger type.

Specifying the Elements. *Default:* None.

pname … One or more pathnames, specifying elements to which the specified trigger type is to be attached.

EXAMPLES

- Attach a trigger to element *hello.c*.

```
% cleartool mktrigger trig1 hello.c
Added trigger "trig1" to attached list of "hello.c".
```

- Attach a trigger to element *util.c*, even if its element type does not appear in the trigger type's restriction list.

```
% cleartool mktrigger –force trig1 util.c
Added trigger "trig1" to attached list of "util.c".
```

- Attach a trigger to directory element *src*.

```
% cleartool mktrigger trig1 src
Added trigger "trig1" to attached list of "src".
Added trigger "trig1" to inheritance list of "src".
```

- Add a trigger to the *release* directory's inheritance list, but not to its attached list.

```
% cleartool mktrigger –nattach trig1 release
Added trigger "trig1" to inheritance list of "release".
```

SEE ALSO

cleartool subcommands: describe, lstype, mktrtype, rmtrigger
profile_ccase

NAME mktrtype – create a trigger type object

SYNOPSIS

- Create element trigger type:

  mktrtype **–ele·ment** [**–glo·bal**] [**–rep·lace**] { **–pre·op** | **–pos·top** } *opkind*[,...]
  [*restriction-list*] [**–nus·ers** *login-name*[,...]]
  { **–exe·c** *command* | **–mkl·abel** *label-type* | **–mka·ttr** *attr-type*=*value*
  | **–mkh·link** *hlink-type*,**to=***pname* | **–mkh·link** *hlink-type*,**from=***pname* } ...
  [**–pri·nt**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  [**–vob** *pname-in-vob*] *type-name* ...

- Create type trigger type:

  mktrtype **–typ·e** [**–rep·lace**] { **–pre·op** | **–pos·top** } *opkind*[,...] *inclusion-list*
  [**–nus·ers** *login-name*[,...]] **–exe·c** *command* [**–pri·nt**]
  [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  [**–vob** *pname-in-vob*] *type-name* ...

- A *restriction-list* contains one or more of:

  –att·ype *attr-type*[,...]
  –brt·ype *branch-type*[,...]
  –elt·ype *elem-type*[,...]
  –hlt·ype *hlink-type*[,...]
  –lbt·ype *label-type*[,...]
  –trt·ype *trigger-type*[,...]
  –rpt·ype *replica-type*[,...]

  NOTE: `-xxtype aaa,bbb` is equivalent to `-xxtype aaa -xxtype bbb`.

- An *inclusion-list* contains any of the components of a *restriction-list*, or one or more of:

  –att·ype *attr-type*[,...] or **–att·ype –all**
  –brt·ype *branch-type*[,...] or **–brt·ype –all**
  –elt·ype *elem-type*[,...] or **–elt·ype –all**
  –hlt·ype *hlink-type*[,...] or **–hlt·ype –all**
  –lbt·ype *label-type*[,...] or **–lbt·ype –all**
  –trt·ype *trigger-type*[,...] or **–trt·ype –all**
  –rpt·ype *replica-type*[,...] or **–rpt·ype –all**

DESCRIPTION

Creates one or more *trigger types* for use within a VOB. A trigger type defines a sequence of one or more *trigger actions* to be performed automatically when a specified ClearCase operation occurs. The set of operations that initiates each trigger action — "causes the trigger to fire" — can be very limited (for example, `checkout` only) or quite general (for example, any operation that modifies an element). You can use a *restriction list* to further limit the circumstances under which a trigger action will be performed.

Only a VOB's owner or the *root* user can create a trigger type.

There are three kinds of trigger types:

- An *element trigger type* works like a label type or attribute type: an instance of the type (that is, a *trigger*) must be explicitly attached to one or more individual elements with the *mktrigger* command. The trigger actions are performed when the specified operation is invoked on any of those elements.
- A variant of the above, called a *global element trigger type*, is associated with the entire VOB. (Hence, no *mktrigger* command is required.) In effect, an instance of the type is implicitly attached to each element in the VOB, even those created after this command is executed.
- A *type trigger type* is associated with one or more type objects. The trigger actions are performed when any of those type objects is modified or used (including creation and deletion of instances of those types).

Trigger Firing

Causing a set of trigger actions to be performed is termed *firing a trigger*. Each trigger action can be:

- Any command (or sequence of commands) that can be invoked from a shell. A command can use special environment variables (EVs), described below, to retrieve information about the ClearCase operation.
- Any of several *built-in* actions defined by *mktrtype*. The built-in actions attach meta-data annotations to the object involved in the ClearCase operation.

Trigger actions execute with the user-ID of the process that caused the trigger to fire.

Interactive Trigger Action Scripts. A script executed as (part of) a trigger action can interact with the user. The *clearprompt* utility is designed for use in such scripts; it can handle several kinds of CLI-style and GUI-style user interactions.

Multiple Trigger Firings. A single ClearCase operation can cause any number of triggers to fire. The firing order of such "simultaneous" triggers is indeterminate. It is also possible for triggers to "chain". For example, a checkin operation might fire a trigger that attaches an attribute to the checked-in version; the "attach attribute" operation might, in turn, fire a trigger that sends mail to an administrator.

If a trigger is defined to fire on a hyperlink operation, and the hyperlink connects two elements, then the trigger will fire twice — once for each end of the hyperlink.

Suppressing Trigger Firing. The firing of a trigger can be suppressed when the associated operation is performed by certain users. Firing of a global element trigger is suppressed if the trigger type has been made obsolete. (See the *lock* manual page).

PRE-OPERATION AND POST-OPERATION TRIGGERS

A *pre-operation trigger* (`–preop` option) fires before the corresponding ClearCase operation begins. The one or more actions you've specified (with `–exec` and/or the options for built-in actions) take place in their order on the command line.

This kind of trigger is useful for enforcing policies:

- If any trigger action returns a non-zero exit status, the ClearCase operation is cancelled.
- If all trigger actions return a zero exit status, the ClearCase operation proceeds.

For example, a pre-operation trigger might prohibit checkin of an element that fails to pass a code-quality test.

A *post-operation trigger* (–`postop` option) fires after completion of the corresponding ClearCase operation. The one or more actions you've specified (with –`exec` and/or the options for built-in actions) take place in their order on the command line. This kind of trigger is useful for recording — in the VOB or in the outside world — the occurrence of the operation. If a post-operation trigger action returns a non-zero exit status, ClearCase displays a `failed exit status` warning message, but continues to perform other trigger actions, if any.

For example, a post-operation trigger on `checkin` might attach an attribute to the checked-in version and send a mail message to interested users and/or managers.

RESTRICTION LISTS AND INCLUSION LISTS

You can define an element trigger type or global element trigger type with a *restriction list*, which limits the scope of the operation specified with –`preop` or –`postop`. The trigger will fire only if the operation involves particular type objects.

A type trigger type is not associated with element objects, but with one or more type objects. When creating a type trigger type, you must specify an *inclusion list*, naming the type objects to be associated with the new trigger type. (Hence, it is unnecessary to use *mktrigger* to create the association.) The special keyword `-all` allows you to associate a type trigger type with *every* type object of a particular kind (for example, all branch type objects), even those objects created after you enter this command.

TRIGGER ENVIRONMENT VARIABLES

When a trigger fires, the trigger action executes in a special environment whose EVs make information available to –`exec` routines: what operation caused the trigger to fire, what object was involved in the operation, and so on. The complete set of EVs is listed in the "Tables" section below.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type creator (applies to –replace only), VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, trigger type (applies to –replace only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Trigger Type. *Default:* None.

–ele·ment Creates an *element trigger type*, which can be attached to individual elements with *mktrigger*.

–ele·ment –glo·bal
Creates a *global element trigger type*, which is effectively (and automatically) attached to the entire VOB.

–typ·e Creates a *type trigger type*, and associates it with specific type objects and/or kinds of type objects.

Specifying the Operation(s) to be Monitored. *Default:* None.

–pre·op *opkind*[,...]
Specifies one or more operations that will cause the trigger to fire *before* the ClearCase operation starts. The exit status of the trigger action(s) is significant: for each trigger action, a zero exit status allows the ClearCase operation to proceed; a non-zero exit status cancels the ClearCase operation.

–pos·top *opkind*[,...]
Specifies one or more operations that will cause the trigger to fire *after* the ClearCase operation completes. The exit status of the trigger action is not significant.

For both `–preop` and `–postop`, you must specify a comma-separated list of ClearCase operations, any of which will cause trigger firing. Many of the operation keywords have the same names as *cleartool* subcommands (for example, `checkout`, `unlock`). Uppercase keywords (for example, MODIFY_ELEM) identify groups of operations. See the "Tables" section below for a list of operation keywords.

Element Trigger Types: Specifying a Restriction List. *Default:* No restrictions — trigger firing will occur when any of the specified operations occurs, no matter what type objects are involved.

–att·ype *attr-type*[,...]
–brt·ype *branch-type*[,...]
–elt·ype *elem-type*[,...]
–hlt·ype *hlink-type*[,...]
–lbt·ype *label-type*[,...]
–trt·ype *trigger-type*[,...]
–rpt·ype *replica-type*[,...]
Use one or more of the above options (or multiple options of the same kind) to specify a set of type objects for the *restriction list*. The type objects must already exist. Repeated options, such as `–elt text_file –elt c_source`, are equivalent to a single option: `–elt text_file,c_source`. Wildcarding (`-eltype '*file'`) is not supported.

At trigger firing time, the items on the restriction list form a logical condition. If the condition is met, the trigger fires; otherwise, the trigger does not fire. (NOTE: Suppressing the firing of a pre-operation trigger means that the ClearCase operation is allowed to proceed.) Here is a simple condition:

`–brtype rel2_bugfix`
Fire the trigger only if the operation involves a branch of type *rel2_bugfix*.

If the list includes multiple type objects, they are combined into a compound condition: type objects of the same kind are grouped with logical OR; objects (or groups) of different kinds are then logically ANDed.

`–brtype rel2_bugfix –eltype text_file,c_source`
Fire the trigger only if the operation involves a branch of type *rel2_bugfix* AND it involves either an element of type *text_file* OR of an element of type *c_source*.

In forming the condition, a type object is ignored if it could not possibly be affected by the ClearCase operation. (The relevant information is included in the "Tables" section below.) For example, the restriction list `-lbtype REL2,REL2.01` applies only to the ClearCase operations *chtype*, *mklabel*, and *rmlabel*.

Type Trigger Types: Specifying an Inclusion List. *Default:* None — you must specify at least one item for the inclusion list of a type trigger type.

–att·ype *attr-type*[,...] or **–att·ype –all**
–brt·ype *branch-type*[,...] or **–brt·ype –all**
–elt·ype *elem-type*[,...] or **–elt·ype –all**
–hlt·ype *hlink-type*[,...] or **–hlt·ype –all**
–lbt·ype *label-type*[,...] or **–lbt·ype –all**
–trt·ype *trigger-type*[,...] or **–trt·ype –all**
–rpt·ype *replica-type*[,...] or **–rpt·ype –all**
You must specify at least one existing type object, or at least one kind of type object, using the special keyword `–all`. The trigger fires only if the inclusion list contains the type object that is being modified or used by the ClearCase operation.

Handling of Name Collisions. *Default:* An error occurs if a trigger type named *type-name* already exists in the VOB.

–rep·lace Replaces the existing definition of *type-name* with a new one. You must specify *all* options, even those that you wish to preserve from the old definition. Additional restriction:

- If an instance of an element trigger type is currently attached to any element, the replacement definition must also be of a (non-global) element trigger type. (You can remove an existing trigger type and all of its attached instances with a command like `cleartool rmtype -trtype -rmall old_Trigger`.)

Suppressing Trigger Firing for Certain Users. *Default:* Trigger firing occurs no matter who performs the ClearCase operation.

–nus·ers *login-name*[,...]
Suppresses trigger firing when any user on the comma-separated *login-name* list performs the operation.

Specifying the Trigger Action. *Default:* None. Specify one or more of the following options to indicate the action to be performed when the trigger fires; you can use more than one option of the same kind. With multiple options, the trigger actions will be performed in the specified sequence.

–exe·c *command*
Executes the specified command in a Bourne shell when the trigger fires. If *command* includes one or more arguments, quote the entire string. Use single-quotes if the command includes ClearCase environment variables, in order to delay interpretation until trigger firing time.

–mkl·abel *label-type*
(with –postop only) Attaches the specified version label to the version involved in the operation that caused trigger firing.

–mka·ttr *attr-type*=*value*
(with –postop only) Attaches the specified attribute name/value pair to the object involved in the operation that caused trigger firing.

–mkh·link *hlink-type*,**to=***pname*
(with –postop only) Creates a hyperlink — *from* the object involved in the operation that caused the trigger to fire, *to* the object specified by *pname*.

–mkh·link *hlink-type*,**from=***pname*
(with –postop only) Creates a hyperlink — *from* the object specified by *pname*, *to* the object involved in the operation that caused the trigger to fire.

NOTES: With the built-in actions –mklabel, –mkattr, and –mkhlink, you can specify the information either literally or using environment variables:

`-mklabel RLS_2.3`	*(literal)*
`-mklabel RLS_$RLSNUM`	*(depends on value of EV at trigger firing time)*
`-mklabel $THIS_RLS`	*(depends on value of EV at trigger firing time)*
`-mkattr ECO=437`	*(literal)*
`-mkattr ECO=$ECONUM`	*(depends on value of EV at trigger firing time)*

The built-in actions never cause additional triggers to fire. But scripts invoked with –exec may cause such "chaining" to occur. For example, a *mklabel* command in a shell script can cause another trigger to fire, but the corresponding –mklabel trigger action cannot.

Tracing Trigger Execution. *Default:* At trigger firing time, if environment variable CLEARCASE_TRACE_TRIGGERS is set to a non-null value in the process that causes the trigger to fire: (1) A message that includes the trigger type name is sent to *stdout* when the trigger fires; (2) A similar message is generated when the trigger action completes.

–pri·nt Causes the messages to be generated at trigger firing time, whether or not CLEARCASE_TRACE_TRIGGERS is set.

VOB Specification. *Default:* The trigger type is created in the VOB that contains the current working directory.

–vob *pname-in-vob*
Specifies the VOB in which to create the trigger type(s). *pname-in-vob* can be the pathname of any object within the VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –cqe). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Naming the Trigger Type. *Default:* None.

type-name …

One or more names for the trigger types to be created. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

TABLES: TRIGGER OPERATIONS AND TRIGGER ENVIRONMENT VARIABLES

Trigger Operations for 'Type' Trigger Types

Table 1 lists the *opkind* keywords for use in definitions of type trigger types (`mktrtype -type`).

Table 1. Operation Keywords for 'Type' Trigger Types

MODIFY_TYPE
mktype (see NOTE)
rmtype
rntype
lock
unlock
chevent

NOTE: If you specify `mktype`, the corresponding inclusion list cannot specify individual type objects; all relevant options must use the `-all` keyword. For example,

```
... -postop mktype -eltype -all -brtype -all ...
```

Trigger Operations for 'Element' and 'Global Element' Trigger Types

Table 2 lists the *opkind* keywords for use in definitions of element trigger types (`–element` and `–element -global`). See also the *events_ccase* manual page.

Table 2. Operation Keywords for 'Element' and 'Global Element' Trigger Types

Operation Keyword	Restrictions Checked When Trigger Fires
MODIFY_ELEM	
checkout	element type, branch type
reserve	element type, branch type
uncheckout	element type, branch type
unreserve	element type, branch type
MODIFY_DATA	
checkin	element type, branch type
chevent	<see NOTE>
chtype	all type objects
lnname	element type, branch type
lock	<see NOTE>
mkbranch	element type, branch type
mkelem	element type
mkslink	N/A
rmbranch	element type, branch type
rmelem	element type
rmname	N/A
rmver	element type, branch type
unlock	<see NOTE>
MODIFY_MD	
chevent	<see NOTE>
mkattr	element type, attribute type, branch type
mkhlink	element type, hyperlink type, branch type
mklabel	element type, label type, branch type
mktrigger	element type, trigger type
rmattr	element type, attribute type, branch type
rmhlink	element type, hyperlink type, branch type
rmlabel	element type, label type
rmtrigger	element type, trigger type

NOTE: The operation fires a trigger only if the affected object is:

- a branch object or version object (in this case, only element type and branch type restrictions apply)
- an element object (in this case, only element type restrictions apply)
- a type object (in this case, only restrictions on that kind of type object apply)

Trigger Environment Variables

Table 3 lists the EVs that are set in the environment in which a trigger action script runs. The words in parentheses at the beginning of the description indicate which ClearCase operations cause the EV to be set to a significant string; for all other operations, the EV is set to the null string.

Table 3. Trigger Environment Variables

CLEARCASE_ATTACH
(mktrigger, rmtrigger) Set to 1 if a non-global element trigger type is on the affected element's *attached list*; Set to 0 if it is on a directory element's *inheritance list*. See the *mktrigger* manual page for a description of these lists.

CLEARCASE_ATTYPE
(all operations that can be restricted by attribute type) Attribute type involved in operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed attribute type object.

CLEARCASE_BRTYPE
(all operations that can be restricted by branch type) Branch type involved in the operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed branch type object.

CLEARCASE_CI_FPN
(checkin) Pathname in `checkin -from`.

CLEARCASE_COMMENT
(all operation kinds that support comments) Comment string for the command that caused the trigger to fire.

CLEARCASE_ELTYPE
(all operations that can be restricted by element type) Element type of the element involved in the operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed element type object.

CLEARCASE_FTEXT
(mkhlink, rmhlink) Text associated with hyperlink *from-object*.

CLEARCASE_FVOB_PN
(mkhlink, rmhlink) Pathname of VOB containing hyperlink *from-object*.

CLEARCASE_FXPN
(mkhlink, rmhlink) VOB-extended pathname of hyperlink *from-object*.

CLEARCASE_HLTYPE
(all operations that can be restricted by hyperlink type) Hyperlink type involved in operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed hyperlink type object.

CLEARCASE_ID_STR
(checkin, checkout, mkattr, mkbranch, mkhlink, mklabel, rmattr, rmhlink, rmlabel, rmver) *Version-ID* of version, or *branch pathname* of branch, involved in the operation.

CLEARCASE_IS_FROM
(mkhlink, rmhlink) Set to 1 if CLEARCASE_PN contains name of hyperlink *from-object*; set to 0 if CLEARCASE_PN contains name of hyperlink *to-object*.

CLEARCASE_LBTYPE
(all operations that can be restricted by label type) Label type involved in the operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed label type object.

CLEARCASE_MTYPE
(all) Kind of type object involved in the operation that caused the trigger to fire: `element type`, `branch type`, and so on.

CLEARCASE_NEW_TYPE
(rntype) New name of the renamed type object.

CLEARCASE_OP_KIND
(all) Operation that caused the trigger to fire.

CLEARCASE_OUT_PN
(checkout) Pathname in `checkout -out`. (Same as CLEARCASE_PN if `–out` not used.)

CLEARCASE_PN
(all operations; element triggers only) Name of element, as it was specified in the command that caused the trigger to fire.

CLEARCASE_PN2
(lnname)

- When a side-effect of a `mkelem` operation, gets the same value as CLEARCASE_PN.
- When a side-effect of a `mv` operation, gets the old pathname of the element.
- For creation of a VOB hard link (*ln* command), gets the already-existing pathname.

CLEARCASE_POP_KIND
(mkelem, mslink, lnname) Parent operation kind. The `mkelem` and `mslink` operations both cause an `lnname` operation. If `lnname` happens as a result of either of these "parent" operations, CLEARCASE_POP_KIND is set to `mkelem` or `mkslink`, respectively. Note that both the "parent" operations (`mkelem` and `mkslink`) and the "child" operation (`lnname`) set CLEARCASE_POP_KIND to the applicable parent operation value — `mkelem` or `mkslink`.

CLEARCASE_PPID
(all) Parent Process-ID: the process-ID of the ClearCase client program (for example, *cleartool*) that invoked the trigger. This is useful for constructing unique names for temporary files that will pass data between a pre-operation trigger and a post-operation trigger, or between successive parts of a multipart trigger action.

CLEARCASE_RPTYPE
(all operations that can be restricted by replica type) Replica type involved in the operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed replica type object.

CLEARCASE_SLNKTXT
(mkslink; that is, the `ln -s` command) Text of the new VOB symbolic link.

CLEARCASE_TRTYPE
(all operations that can be restricted by trigger type) Trigger type involved in the operation that caused the trigger to fire. In an *rntype* operation, the old name of the renamed trigger type object.

CLEARCASE_TTEXT
(mkhlink, rmhlink) Text associated with hyperlink *to-object*.

CLEARCASE_TVOB_PN
(mkhlink, rmhlink) Pathname of VOB containing hyperlink *to-object*.

CLEARCASE_TXPN
(mkhlink, rmhlink) VOB-extended pathname of hyperlink *to-object*.

CLEARCASE_USER
(all) The user who issued the command that caused the trigger to fire; derived from the UNIX-level *effective user ID*.

CLEARCASE_VAL
(mkattr) String representation of attribute value for CLEARCASE_ATTYPE (for example, `"Yes"` or `4657`).

CLEARCASE_VIEW_TAG
(all) View-tag of the view in which the operation that caused the trigger to fire took place.

CLEARCASE_VOB_PN
(all) VOB-tag of the VOB whose object was involved in the operation that caused the trigger to fire.

CLEARCASE_VTYPE
(mkattr) Value type of the attribute in CLEARCASE_ATTYPE (for example, `string` or `integer`).

CLEARCASE_XN_SFX
(all) Extended naming symbol (such as `@@`) for host on which the operation took place.

CLEARCASE_XPN
(all operations; element triggers only) same as CLEARCASE_ID_STR, but prepended with CLEARCASE_PN and CLEARCASE_XN_SFX values, to form a complete VOB-extended pathname of the object involved in the operation.

EXAMPLES

Trigger environment variables typically are to be evaluated when the trigger fires, not when you enter the *mktrtype* command. If this is the case, escape the dollar-sign (`$`) character, either by enclosing it in single-quotes or by preceding it with a backslash (\). This escaping is not necessary if you enter the command manually in *cleartool*'s interactive mode (that is, if it is not interpreted by a shell).

- Create an element type named *script*, for use with shell-script files. Then, create a global element trigger type, *chmod_a_plus_x*, that makes newly-created elements of type *script* executable. Convert a view-private file to an element of this type.

```
% cleartool mkeltype –supertype text_file –c "shell script" script
Created element type "script".
% cleartool mktrtype –element –global –postop mkelem –eltype script –nc \
    –exec '/usr/atria/bin/cleartool protect –chmod a+x $CLEARCASE_PN' chmod_a_plus_x
```

```
Created trigger type "chmod_a_plus_x".
% cleartool mkelem –eltype script –ci –nc cleanup.sh
Created element "cleanup.sh" (type "script").
Changed protection on "/usr/hw/src/cleanup.sh".
Checked in "cleanup.sh" version "/main/1".
```

- Create a global element trigger type, in order to run a script each time a *checkin* operation takes place.

```
% cleartool mktrtype –element –global –postop checkin –nc \
     –exec /usr/local/bin/notify notify_admin
Created trigger type "notify_admin".
```

'notify' script:

```
mail jones adm <<!
"notify_admin" Trigger:

checkin of "$CLEARCASE_PN"

version: $CLEARCASE_ID_STR
     by: $CLEARCASE_USER

comment:
$CLEARCASE_COMMENT
!
```

- Create a global element trigger type to monitor checkins of elements of type *c_source*. Firing the trigger runs a test program on the file being checked it, and may cancel the checkin.

```
% cleartool mktrtype –element –global –nc –preop checkin \
     –exec '$CLEARCASE_VOB_PN/scripts/metrics_test $CLEARCASE_PN' \
     –eltype c_source metrics_trigger
Created trigger type "metrics_trigger".
```

Use of environment variable CLEARCASE_VOB_PN causes the test program to be retrieved from a location in the current VOB.

- Create a global element trigger type, in order to attach a version label to each new version created on any element's *main* branch.

```
% cleartool mktrtype –element –global –postop checkin –mklabel REL\$BL_NUM  \
     –nc –brtype main label_it
Created trigger type "label_it".
```

Environment variable BL_NUM determines which version label is to be attached. ClearCase evaluates this EV at trigger firing time, because the $ character is escaped.

- Create a type trigger type, in order to send a mail message each time any new branch type is created.

```
% cleartool mktrtype –type –nc –postop mktype –brtype –all \
     –exec '$CLEARCASE_VOB_PN/scripts/mail_admin' new_branch_trigger
Created trigger type "new_branch_trigger".
```

- Create a type trigger type, in order to monitor the creation of new label types. The trigger script aborts the label-type-creation operation if the specified name does not conform to standards.

```
% cleartool mktrtype –type –nc –preop mktype –lbtype –all \
     –exec '$CLEARCASE_VOB_PN/scripts/check_label_name' check_label_trigger
Created trigger type "check_label_trigger".
```

- Create an element trigger type that, when attached to an element, fires whenever a new version of that element is checked in. Firing the trigger attaches attribute *TestedBy* to the version, assigning it the value of the CLEARCASE_USER environment variable as a double-quoted string.

  NOTE: In this example, the single-quotes (1) preserve the double quotes on the string literal, and (2) suppress environment variable substitution by the shell. ClearCase evaluates the CLEARCASE_USER environment variable at firing time.

```
% cleartool mktrtype –element –postop checkin \
      –c "set attribute to record which user checked in this version" \
      –mkattr 'TestedBy="$CLEARCASE_USER"' trig_who_didit
Created trigger type "trig_who_didit".
```

- Create a global element trigger type that prompts for the source of an algorithm when an element of type *c_source* is created. Firing the trigger executes a script named *hlink_algorithm*, which invokes the *clearprompt* utility to obtain the necessary information. The script then creates a "text only" hyperlink between the newly created element object (for example, *foo.c@@*) and the specified text. The *hlink_algorithm* script is shown immediately after the *mktrtype* command.

```
% cleartool mktrtype –element –global –nc –postop mkelem –eltype c_source \
      –exec '$CLEARCASE_VOB_PN/scripts/hlink_algorithm' describe_algorithm
Created trigger type "describe_algorithm".
```

  hlink_algorithm script:

```
clearprompt text -outfile /usr/tmp/alg.$CLEARCASE_PPID -multi_line \
-def "Internal Design" -prompt "Algorithm Source Document:"

TOTEXT=‘cat /usr/tmp/alg.$CLEARCASE_PPID‘

cleartool mkhlink -ttext "$TOTEXT" design_spec $CLEARCASE_PN$CLEARCASE_XN_SFX

rm /usr/tmp/alg.$CLEARCASE_PPID
```

- Use a post-operation trigger to modify the user-supplied comment whenever a new version is created of an element of type *header_file*.

```
% cleartool mktrtype –element –global –nc –postop checkin –eltype header_file \
      –exec '/usr/local/scripts/hdr_comment' change_header_file_comment
Created trigger type "change_header_file_comment".
```

  hdr_comment script:

```
# analyze change to header file
CMNT=‘/usr/local/bin/analyze_hdr_file $CLEARCASE_PN‘

# append analysis to user-supplied checkin comment
cleartool chevent -append -c "$CMNT" $CLEARCASE_PN‘
```

SEE ALSO

cleartool subcommands: describe, lock, lshistory, lstype, mkattr, mklabel, mkhlink, mktrigger, rmtrigger clearprompt, events_ccase, profile_ccase

NAME mkview – create and register a view

SYNOPSIS

mkview **–tag** *view-tag* [**–tco·mment** *tag-comment*] [**–tmo·de** { **msdos** | **unix** }]
[**–ln** *remote-storage-dir-pname*]
[**–reg·ion** *network-region*]
[**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
view-storage-dir-pname

DESCRIPTION

Creates a new *view* by:

- creating a *view storage directory* at a specified location
- creating a *view-tag*, the name with which the view will be accessed by developers
- placing entries in the network's *view registry files*(*/usr/adm/atria/rgy/view_object* and */usr/adm/atria/rgy/view_tag*)
- starting a *view_server* process on the host where the view storage directory physically resides

A view storage directory is the root of a directory tree whose principal contents are a *view database*, a *config spec*, and a *private storage area*. See the *view* manual page for details.

The *view_server* process implements ClearCase's *transparency* feature by converting standard file names and pathnames (for example, *util.c*) into references to particular versions (for example, *util.c@@/main/6*). The *view_server* also manages *view-private objects*. Although these objects appear to be located in VOB directories, they are actually stored in the view's private storage area.

Text Modes

Operating systems vary in the character sequences they use to terminate lines of text files. Each ClearCase view has a *text mode*, which determines the line terminator sequence for text files in that view. By default, *mkview* creates a view with the "unix" text mode; in such views, the line terminator for text files is a single `<NL>` character.

For more details, see section "Text Files, Cleartext, and a View's Text Mode" in the *mkeltype* manual page.

Activating the View

Creating a view-tag causes an implicit *startview* command, which automatically activates the view on the current host (unless the tag's target network region does *not* include the local host.) This places an entry in the host's *viewroot* directory. (For example, specifying `-tag gamma` creates the entry */view/gamma*.) Once activated, the view can be *set* with the *setview* command; it can also be accessed with view-extended naming. (For additional details, see the *startview*, *view*, and *pathnames_ccase* manual pages.)

Access Permissions

Avoid creating views as the *root* user. This often causes problems with remote access to a view, since the *root* user on one host typically becomes *nobody* (often, user-ID –2) when accessing other hosts.

Your current *umask*(1) setting determines which users can access the view. For example, a umask value of 2 allows anyone to read data in the view, but only you (the view's owner) and others in your group can write data to it — create view-private files, build derived objects, and so on. If your umask value is 22, only you will be able to write data to the new view.

Reconfiguring a View

A view's associated *view_server* process reads a configuration file when it starts up. You can revise this file — for example, to change the size of the view's in-memory cache, or to make the view read-only. See the view_server manual page for details.

Deleting Views

The view storage directory created by this command is the root of a standard directory tree; but a view should be deleted only with the *rmview* command, not with standard *rm*(1). See the *rmview* manual page for details.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the View-Tag. *Default:* None.

–tag *view-tag*

Specifies a name for the view, in the form of a simple file name. This name appears in the local host's file system as a subdirectory of */view*, the *viewroot* directory (for example, as */view/experiment*).

This view-tag applies only to the local host's network region. If your network has multiple regions, use the *mktag* command to create an additional view-tag for each additional region.

–tco·mment *tag-comment*

Adds a comment to the view-tag's entry in the *view_tag* registry file. Use `lsview -long`, or the graphical view-tag browser, to display the tag comment.

Specifying a Network Region. *Default:* Creates the view-tag in the local host's network region, which is listed in file */usr/adm/atria/rgy/rgy_region.conf*. See the *registry_ccase* manual page for a discussion of *network regions*.

–reg·ion *network-region*

Creates the view-tag in the specified *network region*. An error occurs if the region does not already exist.

CAUTION: The view-tag created with *mkview* must be for the network region to which the view host belongs. Thus, you should use this option only when you are logged in to a remote host that is in another region. Moreover, a view-tag for the view's "home region" must always exist.

Specifying the Text Mode. *Default:* A *unix mode* view is created; the line terminator for text files is a single `<NL>` character.

–tmo·de unix
Same as default.

–tmo·de msdos
Creates an *msdos mode* view; the line terminator for text files is a `<CR><NL>` sequence.

Remote Private Storage Area. *Default:* Creates the view's *private storage area* as an actual subdirectory of *view-storage-dir-pname*. This subdirectory, named *.s*, will hold checked-out versions, newly-created derived objects, and other view-private objects.

–ln *remote-storage-dir-pname*
Creates the *.s* directory at another location, *remote-storage-dir-pname*. A UNIX-level symbolic link to *pname* is created at *view-storage-dir-pname**/.s***, providing access to the remote storage area. Restrictions:

- *remote-storage-dir-pname* must be a valid pathname on every host (no matter what its network region) from which users will access the view.
- This view cannot be used to export a VOB to a non-ClearCase host. (See the *exports_ccase* manual page.)
- Some operations performed by the *root* user in this view may fail. This is another symptom of the *root*-becomes-*nobody* problem explained in the "Description" section.

This mechanism is independent of the ClearCase *network storage registry* facility. This, the pathname to a remote storage area must be truly global, not just global within a particular network region.

Specifying the View's Location. *Default:* None — you must specify a location for the new view storage directory. The host on which this directory physically resides is termed the *view host*. Using the *view-storage-dir-pname* argument, *mkview* heuristically derives the hostname, local access path, and global access path information for the view; it stores this information in the network's view registry.

An unusual network topology and/or a nonstandard network interface may defeat the heuristic, preventing access by some or all users. In such cases, set the view's registry information explicitly with the `–host`, `–hpath`, and `–gpath` options.

view-storage-dir-pname
The location at which a new view storage directory is to be created: full pathname, relative pathname, or simple subdirectory name. (An error occurs if something already exists at this pathname.) You can create a view storage directory at any location in the file system where the standard UNIX permissions allow you to create a subdirectory, except that:

- You cannot create a view storage directory within a VOB, within another view, or within the viewroot directory.
- *view-storage-dir-pname* must specify a location on a host where ClearCase has been installed. This follows from the fact that the view database files must physically reside on a ClearCase host, to enable access by the *view_server* process.

–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*

These options must appear as a set. Use them only when required to explicitly set a view's registry information. You can use these options when creating a new view, or to update the view registry information for an existing view. You must specify the location of the view storage directory in two ways:

- **Host-local pathname** — The name of a host (`–host`), along with a standard full pathname (`–hpath`) to the desired storage location that is valid on that host. Together, these constitute a *host-local pathname*, which is reported by some commands in a colon-separated format:

  ```
  host3:/view_store/view5.vws
  ```

- **Global pathname** — A standard full pathname (`–hpath`) to the desired storage location. This pathname must be valid on *all* hosts (in the view-tag's network region) from which the view will be accessed, including the host where the view storage directory resides. For example:

  ```
  /net/host3/view_store/view5.vws
  ```

EXAMPLES

- Create a view storage directory and assign it the view-tag *mainRel2*.

  ```
  % cleartool mkview –tag mainRel2 /net/host3/view_store/mainRel2.vws
  Created view.
  Host-local path: host3:/view-store/mainRel2.vws
  Global path:     /net/host3/view-store/mainRel2.vws
  It has the following rights:
  User : anne     : rwx
  Group: dev      : rwx
  Other:          : r-x
  ```

- Create a view storage directory named *Rel2.vws* in the current working directory, but with its private storage area on a remote host.

  ```
  % cleartool mkview –tag Rel2 –ln /net/host4/priv_view_store/Rel2.vps Rel2.vws
  Created view.
  Host-local path: host3:/view-store/Rel2.vws
  Global path:     /net/host3/view-store/Rel2.vws
  It has the following rights:
  User : anne     : rwx
  Group: dev      : rwx
  Other:          : r-x
  ```

- Create a view on the local host, then activate the view on a remote host.

  ```
  % cleartool mkview –tag anneRel2 /view_store/anneRel2.vws
  Created view.
  Host-local path: host3:/view-store/anneRel2.vws
  Global path:     /net/host3/view-store/anneRel2.vws
  It has the following rights:
  User : anne     : rwx
  Group: dev      : rwx
  ```

```
Other:          : r-x
```

```
% rsh host4 cleartool startview anneRel2
```

The remote shell command is named *remsh* on some systems.

SEE ALSO

cleartool subcommands: lsview, rmview, mktag, rmtag, setview, startview, unregister exports_ccase, registry_ccase, type_manager, pathnames_ccase, view, view_server umask(1)

NAME mkvob – create and register a versioned object base (VOB)

SYNOPSIS

mkvob **–tag** *vob-tag* [**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–tco·mment** *tag-comment*]
[**–opt·ions** *mount-options*] [**–pub·lic** [**–pas·sword** *tag-registry-password*]]
[**–reg·ion** *network-region*]
[**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
vob-storage-dir-pname

DESCRIPTION

Creates a new *versioned object base*, or *VOB*, by:

- creating a *VOB storage directory* at a specified location
- creating a *VOB-tag*, which specifies the VOB's mount point — the pathname at which the VOB will be accessed by users
- placing entries in the network's *VOB registry* files (*/usr/adm/atria/rgy/vob_object* and */usr/adm/atria/rgy/vob_tag*)
- starting a *vob_server* process on the host where the VOB storage directory physically resides. (Other server processes for the VOB are started on that host, as needed, when developers start using the VOB.)

A VOB storage directory is the root of a directory tree whose principal contents are a *VOB database* and a set of *storage pools*. See the *vob* manual page for details.

mkvob creates exactly one VOB-tag for the newly-created VOB. This tag applies to the local host's network region. To make additional VOB-tags for other regions, use the *mktag* command. In general, the VOB-tags for a given VOB should all be *public*, or all *private*.

PUBLIC AND PRIVATE VOBS

ClearCase supports the notion that some VOBs are to be shared, while others are to be used only by their creators. Accordingly, there are two kinds of VOB-tags: *public* and *private*.

Public VOB-Tags

A public VOB-tag specifies a location at which any user can mount the VOB. Furthermore, once a public VOB is mounted on a host, any user on that host can access it (subject to the standard access permissions).

Typically, all public VOBs are mounted automatically at ClearCase startup time with the command `cleartool mount –all`. (To create a public VOB that is not mounted automatically, specify `–options noauto` in the *mkvob* command.)

When creating a public VOB-tag with *mkvob* or *mktag*, you must supply the network's *VOB-tag password*; if you don't use the `–password` option, you are prompted to type one. See the *registry_ccase* manual page for information on how the password is stored.

You need not create a public VOB's mount-over directory; the `cleartool mount` command creates it automatically, if necessary.

Private VOB-Tags

A private VOB-tag specifies a mount point at which only the VOB's owner (usually, its creator) can mount the VOB using *cleartool* — for example:

```
cleartool mount /vobs/myPrivateVob
```

The *root* user can use the standard UNIX *mount*(1M) command to bypass the "owner only" mount restriction. The command `cleartool mount -all` does not mount private VOBs.

Once a private VOB is mounted, any user can access it (subject to the standard access permissions). You must explicitly create the mount-over directory for a private VOB; the `cleartool mount` command does not create it automatically.

Private-to-Public Conversion

To convert a private VOB to a public VOB, use a command like this:

```
% cleartool mktag –vob –tag /vobs/vob3.p –replace –public /usr/vobstore/private3.vbs
```

This replaces the VOB's private VOB-tag with a public one. *mktag* prompts you to enter the VOB-tag password.

ACTIVATING THE VOB

A VOB cannot be used for development work on a host until it is *activated* with the `cleartool mount` command. This causes the VOB's storage directory to be mounted on the host at the VOB-tag location, as a file system of type *MVFS*. See the ClearCase *mount* manual page for details.

AUTOMATICALLY-CREATED DIRECTORY ELEMENTS

mkvob automatically creates the following directory elements in a new VOB:

- **VOB root directory** — A *mkdir* command is implicitly executed to create a directory element, the *VOB root directory*, in the new VOB. Activating a VOB makes its root directory accessible at the pathname specified by the VOB-tag (that is, at the VOB mount point).
- **'lost+found' directory** — *mkvob* also creates a special directory element, *lost+found*, as a subdirectory of the VOB root directory. ClearCase places elements that are no longer entered in any directory version in this special directory.

See the *vob* manual page for more information on these directories.

DEFAULT STORAGE POOLS

Each VOB storage directory is created with three default *storage pool* subdirectories:

sdft	default source storage pool
cdft	default cleartext storage pool
ddft	default derived object storage pool

See the *vob* manual page for details.

ACCESS PERMISSIONS

In considering access permissions, it is important to distinguish these two "top-level" directories:

- **VOB storage directory** — the standard directory created by this command, which is actually at the top level of a simple directory tree.
- **VOB root directory** — the ClearCase directory element accessed at the VOB-tag (VOB mount point).

When you create a VOB, your operating system-level UID and GID are assigned to the VOB storage directory and the default storage pools. The mode of the VOB storage directory is set according to your current *umask*(1) setting. This affects which users, and which views, will be able to access the VOB. The modes of storage pool directories are set to 755, regardless of your current umask setting.

WARNING: Do not use standard permission-setting utilities (for example, *chown*(1) or *chgrp*(1)) on a VOB storage directory. This will create inconsistencies and cause confusion.

The mode of the VOB root directory, by contrast, *is* derived from your current *umask*(1) setting. The mode can changed subsequently with the *protect* command. Note that the `w` permission on this directory (as on any directory element) affects only the creation of view-private objects; changes to the VOB itself are controlled by ClearCase-level permissions (see below), not those at the operating system level.

ClearCase-Level Access Permissions

ClearCase implements its own access scheme that goes beyond the standard operating system facilities. When you create a VOB, you become its *VOB owner* (in effect, the "superuser" for the VOB), and your group(s) become its *group list*. These settings control access to many ClearCase operations involving the VOB; the settings can be changed subsequently with the *protectvob* command.

The *protect* command affects access to individual elements and shared derived objects.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the VOB-Tag. *Default:* None.

–tag *vob-tag*

A standard full pathname, which specifies the *mount-over* directory at which the VOB will be mounted as a file system of type MVFS. The VOB-tag is entered in the network's *vob_tag* registry file (*/usr/adm/atria/rgy/vob_tag*).

This VOB-tag applies only to the local host's network region. If your network has multiple regions, use the *mktag* command to create an additional VOB-tag for each region.

If you are creating a private VOB (no `–public` option), you must also create the mount-over directory on each host where you will mount the VOB. (The ClearCase *mount* command automatically creates mount-over directories for public VOBs.)

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–cqe`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

−tco·mment *tag-comment*
Adds a comment to the VOB-tag's entry in the *vob_tag* registry file. Use `lsvob -long` to display the tag comment.

Specifying Mount Options. *Default:*

−opt·ions *mount-options*
(root user only) Options to be used in mounting the VOB. The following options are valid:

ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **noauto**, **nodev**, **nodnlc**, **nosuid**, **retrans**, **timeo**, **acdirmin**, **acdirmax acregmin**, **acregmax**, **actimeo**

See the appropriate operating system manual page (for example, *mount*(1M)) for the meanings of these options. If the mount options list contains white space, enclose it in quotes.

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` mount option.

Public vs. Private VOB. *Default:* Creates a private VOB.

−pub·lic Creates a public VOB. See "Public and Private VOBs" above.

−pas·sword *tag-registry-password*
A password is required to create a public tag. If a `mkvob -public` command line does not include the VOB-tag password, *mkvob* prompts for it. The password is checked against the file */usr/adm/atria/rgy/vob_tag.sec* (see the *registry_ccase* manual page); an error occurs if there is no match. Note that the VOB *does* get created in this case, but without a VOB-tag. Use *mktag* to supply a public or private VOB-tag.

Specifying a Network Region. *Default:* Creates the VOB-tag in the local host's network region, which is listed in file */usr/adm/atria/rgy/rgy_region.conf*. See the *registry_ccase* manual page for a discussion of *network regions*.

−reg·ion *network-region*
Creates the VOB-tag in the specified *network region*. An error occurs if the region does not already exist.

CAUTION: The VOB-tag created with *mkvob* must be for the network region to which the VOB host belongs. Thus, you should use this option only when you are logged in to a remote host that is in another region. Moreover, a VOB-tag for the VOB's "home region" must always exist.

Specifying the VOB's Location. *Default:* None — you must specify a location for the new VOB storage directory. The host on which this directory physically resides is termed the *VOB host*. Using the *vob-storage-dir-pname* argument, *mkvob* heuristically derives the hostname, local access path, and global access path information for the VOB; it stores this information in the network's VOB registry.

An unusual network topology and/or a nonstandard network interface may defeat the heuristic, preventing access by some or all users. In such cases, set the VOB's registry information explicitly with the –host, –hpath, and –gpath options.

vob-storage-dir-pname
The location at which a new VOB storage directory is to be created: full pathname, relative pathname, or simple subdirectory name. (An error occurs if something already exists at this pathname.) You can create a VOB at any location where the operating system allows you to create a subdirectory, except that:

- You cannot create a VOB within an existing VOB storage directory.
- You cannot create a VOB under an existing VOB-tag (VOB mount point).
- You cannot create a VOB within the *viewroot* directory (*/view*).
- *vob-storage-dir-pname* must specify a location on a host where ClearCase has been installed. This follows from the fact that the VOB database (located in subdirectory *db* of the VOB storage directory) must physically reside on a ClearCase host, where it is accessed by ClearCase server programs (*vob_server*, *db_server*, and *vobrpc_server*) running locally.

–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*
These options must appear as a set. Use them only when required to explicitly set a VOB's registry information. You can use these options when creating a new VOB, or to update the registry information for an existing VOB. You must specify the location of the VOB storage directory in two ways:

- **Host-local pathname** — The name of a host (–host), along with a standard full pathname (–hpath) to the desired storage location that is valid on that host. Together, these constitute a *host-local pathname*, which is reported by some commands in a colon-separated format:

  ```
  host2:/usr/vobstore/vob2.vbs
  ```

- **Global pathname** — A standard full pathname (–gpath) to the desired storage location. This pathname must be valid on *all* hosts on which the VOB will be accessed, including the host where the VOB storage directory resides.

EXAMPLES

- Create a private VOB storage directory, *project3.vbs*, in the */usr/vobstore* directory on local host *venus*, and give it the VOB-tag */vobs/project3*. Then, mount the VOB on the local host.

```
% cleartool mkvob –tag /vobs/project3 –c "main development sources" /usr/vobstore/project3.vbs
Created versioned object base.
Host-local path: venus:/usr/vobstore/project3.vbs
Global path:     /net/venus/usr/vobstore/project3.vbs
VOB ownership:
  owner anne
  group dev
```

```
Additional groups:
  group usr
  group adm
% mkdir /vobs/project3                    (create VOB mount point to match the VOB–tag)
% cleartool mount /vobs/project3          (mount VOB as file system of type MVFS)
```

- Create a public VOB, which will be mounted automatically at ClearCase startup time (by all hosts in the current host's network region).

```
% cleartool mkvob –tag /vobs/src1 –public –password tagPword /vobstore/src1.vbs
Created versioned object base.
Host-local path: saturn:/vobstore/src1.vbs
Global path:     /net/saturn/vobstore/src1.vbs
  .
  .
  .
```

- Create a public, read-only VOB that will *not* be mounted automatically at ClearCase startup time (or whenever `cleartool mount -all` executes). Supply the VOB-tag password interactively.

```
% cleartool mkvob –tag /vobs/r1 –public –options ro,noauto /vbs/r1.vbs
Vob tag registry password: <xxx>       (password matches contents of /usr/adm/atria/rgy/vob_tag.sec)
Created versioned object base.
  .
  .
  .
```

SEE ALSO

cleartool subcommands: cd, chpool, lshistory, mkpool, mount, protect, rmelem, rmname, rmvob, uncheckout, umount
crontab_ccase, export_mvfs, exports_ccase, filesys_ccase, mount_mvfs, profile_ccase, registry_ccase, vob
mount(1M), umask(1)

NAME mount – activate a VOB at its VOB-tag directory

SYNOPSIS

- Mount a single VOB:

  mount [**–opt·ions** *mount-options*] *vob-tag*

- Mount all public VOBs:

  mount –a·ll

DESCRIPTION

Prerequisite: The VOB being activated must already have a 'VOB-tag' in the network's 'vob_tag' registry file. See the 'mkvob' and 'mktag' manual pages.

Activates one or more VOBs on the local host by performing UNIX-level *mounts* of their *VOB storage directories*. The *mount* command invokes a short-lived *mntrpc_server* process on the VOB host; this process, running as *root*, performs the actual mount. A VOB is mounted as a file system of type MVFS (ClearCase's *multiversion file system* type).

Mounting All VOBs

The *root* user can use `cleartool mount -all` to mount all *public* VOBs listed in the VOB registry. This command executes at ClearCase startup time — see the *init_ccase* manual page. (It does *not* mount VOBs whose tag entries include the mount option `noauto`.)

Mounting of Public and Private VOBs

A public VOB can be mounted by any user; if the *mount-over* directory does not already exist, it is created automatically.

A private VOB can be mounted with `cleartool mount` only by its owner. The *root* user can use the standard *mount*(1M) command to mount a private VOB; other users cannot mount it at all. The mount-over directory must already exist.

Only the *root* user can use `–options` to specify mount options on the command line, or use `–all` to mount all public VOBs.

See the *mkvob* manual page for a discussion of *public* and *private* VOBs.

VOB-TAGS AND THE VOB STORAGE REGISTRY

You reference a VOB by its *VOB-tag* (the full pathname of its mount point), not by its storage area pathname. The *mount* command uses the VOB-tag to retrieve all necessary information from the *VOB registry*: pathname of VOB storage area, pathname of mount point, and mount options.

VOB Registry vs. File System Table

The VOB registry is a new feature in ClearCase Release 2.0. It is intended to hold *all* information pertaining to the network's ClearCase file systems. If you are upgrading from a previous ClearCase release, we recommend that:

- you remove ClearCase-related information from hosts' standard file system tables
- you discontinue use of the ClearCase-specific file system table, */etc/fstab.mfs*

For compatibility, use of these file system tables is still supported in Release 2.0. Support will be withdrawn in future releases, however.

PERMISSIONS AND LOCKS

Permissions Checking: See "Mounting of Public and Private VOBs" above. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

Specifying Mount Options. *Default:* Mounts each VOB using the `-options` field in its *vob_tag* registry file.

–opt·ions *mount-options*
(*root* user only; mutually exclusive with `–all`) Ignores the `-options` field in the *vob_tag* registry file entry and uses the specified set of options, which can include:

ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **noauto**, **nodev**, **nodnlc**, **nosuid**, **retrans**, **timeo**, **acdirmin**, **acdirmax acregmin**, **acregmax**, **actimeo**

See the appropriate operating system manual page (for example, *mount*(1M)) for the meanings of these options. Enclose this argument in quotes if it contains white space.

If you don't specify a timeout or retransmission option, a default value is used:

timeo 5 seconds
retrans 7 retries

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` mount option.

Specifying the VOB(s). *Default:* None.

vob-tag Mounts the VOB with this *VOB-tag*, which must be specified exactly as it appears in the *vob_tag* registry file.

–a·ll (*root* user only; mutually exclusive with `–options`) Mounts all public VOBs listed in the VOB registry, using the mount options in their *vob_tag* registry entries. (Including the mount option `noauto` in a VOB–tag's registry entry prevents the VOB from being mounted by `mount -all`.)

IMPLEMENTATION NOTE

mount calls the standard *mount*(1M) command, which in turn calls the ClearCase–supplied utility *mount_mvfs* to perform the actual work.

EXAMPLES

- Mount the VOB storage directory that is registered with VOB-tag */vobs/Rel4*.

```
% cleartool mount /vobs/Rel4
```

- Mount all VOBs registered with public VOB-tags.

```
% su                                  (become 'root' user)
# cleartool mount -all                (mount all public VOBs)
```

SEE ALSO

cleartool subcommands: umount, lsview, lsvob, register, mkview, mkvob, mktag init_ccase, mount_mvfs, registry_ccase, mount(1M)

NAME mv – move or rename an element or VOB link

SYNOPSIS

- Rename:

  mv [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname target-pname*

- Move to another directory:

  mv [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* [*pname …*] *target-dir-pname*

DESCRIPTION

NOTE: The directory where the element to be moved/renamed resides must be checked-out. If the target location is another directory, it must be checked-out, also. *mv* automatically appends an appropriate line to the checkout comment for all relevant directories.

The *mv* command relocates — renames or moves — an element or VOB symbolic link. For a file element that is checked-out to your view, it relocates the checked-out version, also. (That is it moves the view-private file with the same name as the element.) If the version is checked-out to *another* view, it just issues a warning:

```
cleartool: Warning: Moved element with checkouts to "overview.doc";
view private data may need to be moved.
```

Moving View-Private Objects

This command is for VOB-database objects; use the standard *mv*(1) command to rename or move view-private files.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* An error occurs if any of the following objects are locked: VOB. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Existing Objects. *Default:* None.

pname One or more pathnames, specifying elements or VOB links. If you specify more than one *pname*, then you must specify a directory (*target-pname*) as the new location.

Specifying the New Location. *Default:* None.

target-pname
The new location for the single element or VOB link specified by *pname*. Both *pname* and *target-pname* must specify locations in the same VOB. An error occurs if an object already exists at *target-pname*.

target-dir-pname
The pathname of an existing directory element, to which the elements or links are to be moved. This directory must be located in the same VOB as the objects being moved.

EXAMPLES

NOTE: In all the examples, all directories involved must be checked out.

- Rename a C-language source file from *hello.c* to *hello_old.c*.

```
% cleartool mv hello.c hello_old.c
Moved "hello.c" to "hello_old.c".
```

- Move all files with a `.c` suffix into the *src* directory.

```
% cleartool mv *.c src
Moved "cm_add.c" to "src/cm_add.c".
Moved "cm_fill.c" to "src/cm_fill.c".
Moved "convolution.c" to "src/convolution.c".
Moved "hello.c" to "src/hello.c".
Moved "hello_old.c" to "src/hello_old.c".
Moved "messages.c" to "src/messages.c".
Moved "msg.c" to "src/msg.c".
Moved "util.c" to "src/util.c".
```

- Rename a symlink from *messages.c* to *msg.lnk*, and show the result with *ls*.

```
% cleartool mv messages.c msg.lnk
Moved "messages.c" to "msg.lnk".
```

```
% cleartool ls –long msg.lnk
symbolic link          msg.lnk --> msg.c
```

SEE ALSO

cleartool subcommands: checkout, cd, ln, ls
profile_ccase

NAME protect – change permissions or ownership of an object

SYNOPSIS

protect [**−cho·wn** *login-name*] [**−chg·rp** *group-name*] [**−chm·od** *permissions*]
[**−fil·e** | **−d·irectory**] [**−r·ecurse**]
[**−c** *comment* | **−cq** | **−cqe** | **−nc**] *pname* …

DESCRIPTION

Sets a UNIX-level attribute — owner, group, or access mode — for one or more elements or shared derived objects. This command is similar to the standard UNIX *chmod*(1), *chown*(1), and *chgrp*(1) commands; but instead of modifying a UNIX-level *inode*, it modifies a VOB database.

The main usage of *protect* is to control access by standard UNIX programs to an element's (or shared derived object's) data. For example, you might make some elements readable by anyone, while making others readable only its group members.

Modifying the access mode of an element changes the access modes of all of its source containers and (if applicable) cleartext containers. That is, the change affects *all* versions, not just the version selected by the current view. There is no way to change the access mode of an individual version.

Some forms of *protect* affect ClearCase-level access. For example, a checkout or checkin is permitted only if the user is the element's owner, or is a member of the element's group.

View-Private Objects

This command affects VOB-database objects only, not view-private objects. For this reason, entering a *protect* command sometimes seems to have no effect:

- Changing an element's protections has no effect on its checked-out version(s). After you *checkin* the element, your view selects the checked-in version, thus making the updated protections appear.
- Changing a DO's protection has no effect on the way the DO appears in the view where it was originally created, or in the view(s) where it has been winked-in. To have your view use a shared DO with updated permissions: (1) use *rm* to remove the DO from your view; (2) use *protect* to change the permissions on the DO in the VOB database; (3) use *clearmake* or the *winkin* command to wink-in the DO, with its new permissions.

You can change the protections on any view-private object (including a checked-out version), with the standard UNIX commands.

A winked-in DO is not really a view-private object, but it behaves like one (so that users in different views can build software independently). Moreover, changing the access mode of a winked-in DO actually *converts* it to a view-private file in your view. See "Manipulating Derived Objects with Standard Commands" in the *derived_object* manual page.

'Owner' Setting

The initial owner of an element is the user who creates it with *mkelem* or *mkdir*. The initial owner of a derived object is the user who builds it with *clearmake*. When the derived object is *winked-in* to another view and becomes shared, its data container is *promoted* to a VOB storage pool. This process preserves the derived object's ownership, no matter who performs the build that causes the wink-in.

See the *ct_permissions* manual page for a list of ClearCase operations that can be performed by an element's owner.

'Group' Setting

The initial group of an element is the principal group of its creator (the group listed in the creator's password entry). The new group specified in a `protect -chgrp` command must be one of the groups on the VOB's *group list*.

See the *ct_permissions* manual page for a list of ClearCase operations that can be performed by members of an element's or derived object's group.

'Read' and 'Execute' Access

The "read" and "execute" access permissions of an element or shared derived object controls access to its data in the standard UNIX manner. The "read" and "execute" permissions specified in a `protect -chmod` command will appear in a standard UNIX *ls* directory listing of the element or shared derived object. The permissions are also applied to all its associated data containers.

NOTE: *protect* sometimes adds "group-read" access to your specification. This ensures that the owner of an element always retains "read" access to its data container(s).

'Write' Access

The meaning of the "write" access permission varies with the kind of object:

- For a file element, "write" access settings are ignored. To obtain write access to a file element, you must check it out.
- For a directory element, "write" access allows view-private files to be created within it. ClearCase-level permissions control changes to the directory element itself, (See the *ct_permissions* manual page).
- For a shared derived object, "write" access allows it to be "overwritten" with a new derived object during a target rebuild. (The shared derived object is not actually affected; rather, the view sees the new, unshared derived object in its place.)

Set-UID and Set-GID Access

You can turn on a file (but not directory) element's *set-UID* bit and/or *set-GID* bit, using either a symbolic argument (for example, `u+s`) or an absolute argument (for example, `4755`). These bits are automatically cleared when a `protect chown` or `protect chgrp` command is entered.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, pool (non-directory elements only). See the "Permissions Checking" section of the *cleartool* manual page.

NOTES: With `protect -chgrp`, you must be a member of the new group, and it must also be in the VOB's group list. Only an element's owner or the *root* user can turn on its *set-UID* bit; only a group member or the *root* user can turn on an element's *set-GID* bit.

OPTIONS AND ARGUMENTS

Specifying Protection Changes. *Default:* None.

–cho·wn *login-name*
New owner for the element(s), in *chown*(1) format. The owner may be either a decimal user ID or a login name found in the *passwd*(4) file.

–chg·rp *group*
New group for the element(s), in *chgrp*(1) format. The group may be either a decimal group ID or a group name found in the *group*(4) file.

–chm·od *permissions*
New access rights — owner, group, other (world) — for the element(s), in standard *chmod*(1) format. Both symbolic and absolute codes are valid, such as `go-x` (symbolic) or `666` (absolute).

Specifying the Object(s). *Default:* None.

–fil·e Restricts the command to changing file elements only.

–d·irectory Restricts the command to changing directory elements only.

pname … One or more pathnames, each of which specifies an element or shared derived object. An extended pathname to a version or branch is valid — but keep in mind that *protect* affects the entire element. Shared derived objects can be referenced by DO-ID.

If you specify multiple *pname* arguments, but you do not have permission to change the protections on a particular object, *protect* quits as soon as it encounters this error.

Processing of Directory Elements. *Default:* Any *pname* argument that specifies a directory causes the directory element itself to be changed.

–r·ecurse Changes the entire tree of elements including and below any *pname* argument that specifies a directory element. VOB symbolic links are not traversed during the recursive descent. (Use `–file` or `–directory` to restrict the changes to one kind of element.)

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Add "read" permission to the file element *hello.c*, for all users.

```
% cleartool protect –chmod +r hello.c
Changed protection on "hello.c".
```

- Change the group ID for all elements in the *src* directory to *user*.

```
% cleartool protect –recurse –chgrp user src
Changed protection on "src".
Changed protection on "src/cm_fill.c".
Changed protection on "src/convolution.c".
Changed protection on "src/hello.c".
```

```
Changed protection on "src/msg.c".
Changed protection on "src/util.c".
```

- Allow users in the same group to "read/write/execute" the shared derived object *hello*, but disable all access by the world. Use an absolute permission specification.

```
% cleartool protect –chmod 770 hello
Changed protection on "hello".
```

SEE ALSO

cleartool subcommands: mkpool, rmelem
ct_permissions, scrubber, profile_ccase
chgrp(1), chmod(1), chown(1), group(4), passwd(4)

NAME protectvob – change owner or groups of a VOB

SYNOPSIS

protectvob [**–for·ce**] [**–cho·wn** *login-name*] [**–chg·rp** *group-name*]
[**–add·_group** *group-name*[,...]] [**–del·ete_group** *group-name*[,...]]
vob-storage-pname ...

DESCRIPTION

Before executing this command, log in to the host where the VOB storage directory resides, as that host's 'root' user. Execute this command only when the VOB is quiescent (no active users); it stops and restarts the associated 'vob_server' process.

protectvob manages the ownership and group membership of the files and directories in a VOB, by changing the OS-level permissions on files and directories within the VOB storage area. If the VOB has remote storage pools, you may need to execute this command on the remote host, as well, in order to complete the permissions update. See "VOBs with Remote Storage Pools" below.

VOB Owner and VOB Group List

A new VOB, created with *mkvob*, takes on the identity of its creator:

- the creator becomes the *VOB owner*
- the creator's principal group becomes the VOB's *principal group*
- the creator's group list becomes the VOB's supplementary *group list*

The *VOB owner* is a privileged user, who can perform almost any operation involving that VOB (in effect, the *superuser* for that VOB). The VOB owner owns all of the VOB's data containers and storage pools. All data container manipulations are performed by a *vob_server* process, which runs with the identity of the VOB owner (see *setuid*(2)).

The VOB's supplementary group list simulates a UNIX feature that enables a user to belong to several groups. (See the *multgrps*(1) manual page.)

Groups and Access Control

The VOB's set of groups controls certain operations:

- **Write access** — A user's *principal* group must be one of the VOB's groups — principal or supplementary — in order for the user to create an element or derived object.
- **Read access** — *Any* of a user's groups must be the VOB's principal group in order for the user to:
  - read a version of a *text_file* element (and any other element type for which cleartext containers are created)
  - perform any other operation that modifies the VOB's data containers (*rmver*, *rmbranch*, *rmelem*, *chpool*, *chtype*, and so on)

In addition, the group of an element or derived object can be changed with `protect -chgrp` only if the new group is on this list.

Access Control at the Individual Object Level

A VOB's *owner* and *group list* are VOB-wide settings. Similar settings are maintained at the individual object level:

- Each element in a VOB has UNIX-level access attributes:
  - user (that is, the element's owner)
  - group (just one, not several)
  - read-write-execute permissions (access mode)

  These attributes control access by standard UNIX programs to the element's data. For example, some elements might be made readable by anyone, while others are made readable only by group members. An element's UNIX-level attributes automatically apply to all of its versions.
- Similarly, UNIX-level access attributes are maintained for each shared derived object in the VOB (whose data container is in a VOB storage pool).

The *protect* command controls the UNIX-level access attributes of elements and shared derived objects. An element's access attributes apply to all its source containers and (if applicable) cleartext containers.

The .identity Directory

The `cleartool -describe -vob` command lists a VOB's owner and its group list. This information is recorded in subdirectory *.identity* of the VOB storage directory. See the *vob* manual page for a description of the contents of this subdirectory.

CAUTION: Do not manipulate the *.identity* directory by any means other than this command. Inconsistent settings will cause ClearCase errors.

VOBs with Remote Storage Pools

If any of a VOB's storage pools physically reside on a remote host (accessed through symbolic links), *protectvob* prompts you to run *protectvob* on the remote host:

```
  ⋮
cleartool: Warning: pool: "/vobstore/vega.vbs/s/s_aux01" is remote.
cleartool: Warning: Login to the remote machine "ccsvr01".
cleartool: Warning: Then run this command again, or run the chown_pool script.
```

This is necessary because running as the *root* user, *protectvob* usually does not have the rights to change permissions in the remote storage directory. In such cases, update each remote host as follows:

1. Log into the host as *root*.
2. Enter the *protectvob* command without specifying any options. (You can specify `–force`, if you wish.) You may need to adjust the *vob-storage-pname* you specify, since it is now a remote location.

In some cases, you may need to run the *chown_pool* script on the remote host, to update one or more individual storage pools there. This script is located in the ClearCase *etc* directory. See "Examples" below.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: root user. *Locks:* An error occurs if any of the following objects are locked: VOB. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Confirmation Step. *Default: protectvob* asks for confirmation before changing the permissions in one or more storage pools.

–for·ce Suppresses the confirmation step.

Changing VOB Ownership. *Default:* None — you can use –chown by itself, or in combination with –chgrp.

–cho·wn *user*
Specifies a new VOB owner. *user* can be either a login name or a numeric user-ID. That user becomes the owner of all the VOB's storage pools and all of the data containers in them.

protectvob rebuilds the *.identity* subdirectory of the VOB storage directory, reflecting the new VOB owner's user-ID, group-ID, and additional groups (if any).

–chg·rp *group*
Specifies a new principal group for the VOB. *group* can be either a group name or a numeric group-ID.

Maintaining the Secondary Group List. *Default:* None — you can use –add_group and –delete_group singly, or together.

–add·_group *group*[,...]
Adds one or more groups to the VOB's secondary *group list*. *group* can be either a group name or a numeric group-ID.

–del·ete_group *group*[,...]
group can be either a group name or a numeric group-ID. NOTE: This option can delete only those groups on the VOB's secondary group list, not the principal group. You must use the –chgrp option to change the principal group.

vob-storage-pname
Pathname of a VOB storage directory.

EXAMPLES

- Make user *jackson* the owner of the VOB whose storage area is */usr/lib/vob.vb*.

```
% cleartool protectvob –chown jackson /usr/lib/vob.vb
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.
Pool "sdft" needs to be protected correctly.
Pool "ddft" needs to be protected correctly.
Pool "cdft" needs to be protected correctly.
Protect versioned object base "/usr/lib/vob.vb"?  [no] yes
Do you wish to protect the pools that appear not to need protection?  [no] no
Protecting "/usr/lib/vob.vb/s/sdft"...
Protecting "/usr/lib/vob.vb/s/sdft/0"...
Protecting "/usr/lib/vob.vb/s/sdft/1"...
...
Protecting "/usr/lib/vob.vb/d/ddft"...
Protecting "/usr/lib/vob.vb/d/ddft/0"...
...
Protecting "/usr/lib/vob.vb/c/cdft"...
```

```
Protecting "/usr/lib/vob.vb/c/cdft/2d"...
Protecting "/usr/lib/vob.vb/c/cdft/35"...
...
VOB ownership:
  owner jackson
  group user
Additional groups:
  group doc
```

- Change the owner and group of a remote VOB storage pool.

  ```
  # rlogin ccsvr01
  Password: <enter password>
  # /usr/atria/etc/chown_pool jackson.user /vobaux/vega_src/s001
  ```

- Add one group to a VOB's group list, and remove another group:

  ```
  % cleartool protectvob –add_group devel –delete_group doc /usr/lib/vob.vb
  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.
  Pool "sdft" appears to be protected correctly.
  Pool "ddft" appears to be protected correctly.
  Pool "cdft" appears to be protected correctly.
  Protect versioned object base "/usr/lib/vob.vb"?  [no] yes
  Do you wish to protect the pools that appear not to need protection?  [no] no
  VOB ownership:
    owner jackson
    group user
  Additional groups:
    group devel
  ```

SEE ALSO

cleartool subcommands: chpool, mkpool, mkvob, protect
albd_server, ct_permissions, vob, vob_server, multgrps(1)
ClearCase Administrator´s Manual

NAME pwd – print working directory

SYNOPSIS

pwd

DESCRIPTION

Lists the current working directory, just like the standard command *pwd*(1). This command is intended for use in interactive *cleartool* sessions, and in shell scripts that simulate interactive sessions.

In version-extended namespace, the current working directory is listed as a pathname that is both view-extended and version-extended. It includes the version of each directory element between the current location and the *VOB root directory*. For example:

```
% cd util.c@@/main
% cleartool pwd
/view/akp@@/usr/hw/main/1/src/main/1/util.c/main
```

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

EXAMPLES

- List the name of the current working directory.

```
cleartool> pwd
/usr/hw
```

- Use a view-extended pathname to go to the */usr/hw/src* directory in the context of the *jackson_old* view, and then list the name of the directory.

```
cleartool> cd /view/jackson_old/usr/hw/src
cleartool> pwd
/view/jackson_old/usr/hw/src
```

- Change to a version-extended namespace directory, and list its name. Then change back to the original directory, and list its name.

```
cleartool> cd src@@
cleartool> pwd
/view/jackson_vu@@/usr/hw/main/2/src
cleartool> cd /usr/hw/src
cleartool> pwd
/usr/hw/src
```

SEE ALSO

cleartool subcommands: cd, pwv

NAME pwv – print working view

SYNOPSIS

pwv [**–s·hort**] [**–wdv·iew**] [**–set·view**]

DESCRIPTION

Lists the view-tag of your current *view context*, or `** NONE **` if there is none. You can establish or change your view context by entering a *setview* command, or by *cd*'ing to a view-extended pathname. If you do both, you have two view contexts:

- Your *working directory view* is used to process simple file names and relative pathnames.
- Your *set view* is used to process full pathnames, which begin with a slash (`/`) character.

If you *cd* to a version-extended pathname, *pwv* adds the extended naming symbol to the view-tag (see "Examples").

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Working Directory View vs. Set View. *Default:* Lists both your working directory view and your set view, unless you specify `–short`.

–wdv·iew Lists your working directory view only.

–set·view Lists your set view only.

Listing Format. *Default:* The annotation `Working directory view:` or `Set view:` precedes a view's view-tag.

–s·hort Omits the annotation string.

Specifying `–short` automatically invokes `–wdview` also, unless you use `–setview`.

EXAMPLES

- List the current set view and working directory view. In this case, they are the same.

```
% cleartool pwv
Working directory view: jackson_vu
Set view: jackson_vu
```

- List the working directory view only.

```
% cleartool pwv –wdview
Working directory view: jackson_vu
```

- List the current view after changing the working directory view, but prior to setting a view.

```
% cd /view/jackson_old/usr/hw/src
% cleartool pwv
Working directory view: jackson_old
Set view: ** NONE **
```

- List the current view after setting a view, and then changing the working directory view.

```
% cd /view/jackson_old/usr/hw/src
% cleartool pwv
Working directory view: jackson_old
Set view: jackson_vu
```

- List the current view after changing to a version-extended namespace directory. Use the short format to list the view name only.

```
% cd src@@
% cleartool pwv –short
jackson_vu@@
```

SEE ALSO

cleartool subcommands: cd, setview, startview
pathnames_ccase

NAME quit – quit interactive cleartool session

SYNOPSIS

q·uit

DESCRIPTION

Ends an interactive *cleartool* session, returning control to the parent process. You can also exit by typing a UNIX EOF character (typically, `<Ctrl-D>`) or by entering the *exit* command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

EXAMPLES

- End a *cleartool* interactive session.

```
cleartool> quit
%
```

- End a *cleartool* interactive session with the *quit* synonym, *exit*.

```
cleartool> exit
%
```

- End a *cleartool* interactive session with the UNIX EOF character.

```
cleartool> <Ctrl–D>
%
```

NAME recoverview – recover a view database

SYNOPSIS

- Repair a view:

  recoverview [**–for·ce**] { **–tag** *view-tag* | *view-storage-dir-pname* }

- Recover files associated with deleted VOB or deleted directory:

  recoverview [**–for·ce**] { **–vob** *vob-identifier* | **–dir** *dir-identifier* }
  { **–tag** *view-tag* | *view-storage-dir-pname* }

DESCRIPTION

Repairs a *view database* and the associated *private storage area*, typically after a system crash or similar mishap. If the view does not require recovery, *recoverview* displays a message and takes no other action.

You may also wish to use this command to regain access to *stranded* view-private files. (See "Recovering View-Private Storage" below.)

Automatic Recovery

When necessary, *recoverview* is invoked automatically by a view's associated *view_server* process. Enter this command yourself if messages in the *view log* (*/usr/adm/atria/log/view_log*) suggest view database corruption (for example, `INTERNAL VIEW DB ERROR`).

Possible Data Loss

recoverview uses *reformatview* — that is, recovery involves a dump/load of the view database. (See the *view* manual page.) (*recoverview* automatically deletes the old, invalid view database, which *reformatview* has renamed to *db.dumped*.)

Depending on the state of the view database, this process may cause certain information to be lost. After a view is recovered, consult */usr/adm/atria/log/view_log* to investigate possible data loss. The *set-UID* bit is *always* lost on files that are not owned by the view's owner. See the *reformatview* manual page for more information.

RECOVERING VIEW-PRIVATE FILES: VIEW LOST+FOUND DIRECTORY

In normal ClearCase operation, a file in view-private storage is accessed through a VOB pathname. That is, the file *seems* to be located in the VOB, but is actually stored in the view. This view-VOB correspondence can be disrupted, however:

- A VOB can become temporarily unavailable — for example, by being unmounted.
- A VOB can become permanently unavailable, by being deleted.
- A particular VOB directory can become permanently unavailable, by being deleted with a *rmelem* command.

In all these cases, view-private files that are accessed through the unavailable VOB structure become *stranded* — the files cannot be used for normal ClearCase operations, since there are no VOB pathnames through which they can be accessed. You can resynchronize your view with the available VOBs with *recoverview*'s `–vob` and `–dir` options. This "recovers" stranded files by moving them into the view's *lost+found*, subdirectory *.s/lost+found* of the view storage directory. Such recovered files remain

inaccessible to normal ClearCase operations; you can access them through the view storage directory, using standard UNIX commands.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Confirmation Step. *Default: recoverview* asks for confirmation before modifying a view.

–for·ce Suppresses the confirmation step.

Specifying the View. *Default:* None.

–tag *view-tag*
The view-tag of any registered view.

view-storage-dir-pname
The pathname of a view storage directory.

CAUTION: Make sure that the current working directory is not the same as, or anywhere below *view-storage-dir-pname*.

Recovering View-Private Storage. The following options take ClearCase-internal identifiers for a VOB or a VOB directory (*vob-identifier* and *dir-identifier*) as arguments. The *lsprivate* command uses these identifiers when listing an inaccessible VOB or VOB directory.

–vob *vob-identifier*
Moves all view-private files that correspond to the specified VOB to the view's *lost+found* directory.

–dir *dir-identifier*
Moves all view-private files that correspond to the specified directory element to the view's *lost+found* directory.

CAUTION: If the VOB or directory is still accessible, using these options is probably incorrect — it will *un*synchronize the view and VOB, not synchronize them.

EXAMPLES

NOTE: *recoverview* writes status messages to the *view_log* file in */usr/adm/atria/log*; it does not print status messages on the standard output device.

- Recover the database of a view whose view-tag is *jackson_fix*.
  ```
  % cleartool recoverview –tag jackson_fix
  ```
- Recover the database of a view whose storage directory is */usr/home/jackson/ccviews/std.vws*.
  ```
  % cleartool recoverview /usr/home/jackson/ccviews/std.vws
  ```

SEE ALSO

cleartool subcommands: reformatview
errorlogs_ccase, view

NAME reformatview – update the format of a view database

SYNOPSIS

reformatview [**–dum·p** | **–loa·d**] { **–tag** *view-tag* | *view-storage-dir-pname* }

DESCRIPTION

Changes the format (*schema*) of a *view database* from the format used in a previous ClearCase release to the current format. A view database is a set of binary files in the view storage directory. A new release may use a different database format in order to support new product features, to enhance storage efficiency, or to improve performance.

View database conversion involves two major steps:

- "Dumping" the existing database to a set of ASCII files. This step invalidates the view database, which is renamed to *db.dumped*. You cannot use the view until its database is reloaded.
- "Loading" the ASCII files into a new database that uses the new format.

  NOTE: This does *not* overwrite the old, invalid view database; it remains in the view storage directory, as *db.dumped*, until you explicitly delete it with a standard operating system command.

A view's *view_server* process automatically detects the need for reformatting, and displays a message to this effect. *reformatview* itself writes status messages to */usr/adm/atria/log/view_log*, not to *stdout* or *stderr*.

You can also use *reformatview* to move a view storage area between hosts of different architectures — that is, hosts on which there are differences in the binary files that implement the view database. See the *ClearCase Administrator´s Manual* for a step-by-step procedure.

Possible Data Loss

If the view database requires recovery, some information may be lost in the dump/load process. For example, if a view-private file is owned by someone other than the owner of the view storage area, *reformatview* always strips its *set-UID* bit (if the bit is set).

In addition, some view-private files may be moved into the view's *lost+found* directory. See the *recoverview* manual page for details.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Forcing a Dump. *Default:* If a view's database does not require reformatting (its schema is up-to-date), *reformatview* displays a message and takes no other action; if the schema is out-of-date, *reformatview* performs a dump, then a load.

–dum·p performs only the first step — creating an ASCII *dump* of the view database in file *view_db.dump_file* in the view storage directory.

–loa·d performs only the second step — replacing the old view database with a new one, using the contents of a previously created ASCII dump file.

Specifying the View. *Default:* None.

–tag *view-tag*
The view-tag of any registered view.

view-storage-dir-pname
The pathname of a view storage directory.

CAUTION: Make sure that the current working directory is not the same as, or anywhere below *view-storage-dir-pname*.

EXAMPLES

- Reformat a view whose view-tag is *jackson_old*.
```
% cleartool reformatview –tag jackson_old
```
- Reformat a view whose storage directory is */home/jackson/ccviews/fix.vws*.
```
% cleartool reformatview /home/jackson/ccviews/fix.vws
```

SEE ALSO

cleartool subcommands: recoverview
errorlogs_ccase

NAME reformatvob – update the format (schema) of a VOB database

SYNOPSIS

reformatvob [**–dum·p** | **–loa·d**] [**–rm**] [**–for·ce**] [**–to** *dumpfile-dir-pname*]
[**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
vob-storage-dir-pname

DESCRIPTION

Always back up a VOB's storage directory before entering this command.

This is a 'one-way' command. It must be allowed to complete both the 'dump' and 'load' phases (though these two phases can take place at different times). You cannot abort and undo a 'reformatvob' operation after you have started it; you can only restart and complete the operation.

reformatvob changes the format (*schema*) of a *VOB database* from a format used in a previous ClearCase release to the current format. A new release may use a different database format in order to support new product features, to enhance storage efficiency, or to improve performance.

You can also use *reformatvob*:

- when moving a VOB storage directory between hosts of different architectures — hosts with different binary formats for the files that implement the VOB database
- to "compact" a VOB database, physically deleting records that have been logically deleted by *vob_scrubber*.

In these latter cases, the VOB database is already using the current revision of the schema, and *reformatvob* locks the VOB before reformatting it. An error occurs if the VOB is already locked.

Dumping and Loading

VOB database reformatting involves two phases:

- "Dumping" the existing VOB database to a set of ASCII files. This phase is performed by the *db_dumper* program located within the VOB storage directory. *reformatvob* evaluates disk-space availability before beginning its work, and displays a message if your disk space seems to be insufficient. See "Working with Limited Disk Space" below.
- "Loading" the ASCII files into a new VOB database that uses the new format. This phase is performed by the *db_loader* program, located in */usr/atria/etc*.

By default, both these phases are performed at the same time. The `–dump` and `–load` options enable you to perform them separately.

Both *db_dumper* and *db_loader* are *setUID-root* programs. An invocation of *reformatvob* will fail if these programs have the wrong ownership or the wrong access mode:

```
cleartool: Error: Database dumper "<VOB-pname>/db.reformat/db_dumper"
must be setUID and owned by the super-user.
```

For more on this topic, see the *db_dumper* manual page.

Registering the VOB (Again)

reformatvob updates (or creates, if necessary) the VOB's entry in the network's *vob_object* registry file. However, *reformatvob* does not affect the *vob_tag* registry file. If the newly reformatted VOB does not have an appropriate VOB-tag registry entry, *reformatvob* displays a message advising you to create one or more VOB–tags with *mktag*. For more information on VOB–tags and registries, see the *mkvob* and *registry_ccase* manual pages.

Migrating from ClearCase Release 1.1.x to Release 2

The *reformatvob* command, like all ClearCase Release 2 commands, operates only on registered VOBs. Therefore, before proceeding, *reformatvob* first registers a Release 1.1.x VOB in the *vob_object* registry file. In general, you need only supply the VOB storage directory pathname, which *reformatvob* uses to create the *vob_object* registry entry and, then, to reformat the VOB. Unless you have already created one or more VOB-tags for the "new" VOB, *reformatvob* displays a message advising you to do so. See also "Updating a Host's Existing ClearCase Data" in the *ClearCase Notebook*, and the *registry_ccase* manual page.

Restrictions

The VOB storage directory must physically reside on the host where you enter this command.

The current working directory must not be at or below the VOB storage directory.

Your shell must not have a view context — neither set view nor working directory view.

Working with Limited Disk Space

Running *reformatvob* can substantially increase the amount of disk space used by the VOB storage directory:

- By default, the old VOB database is preserved in a renamed subdirectory of the VOB storage directory. You can use the `-rm` option to discard the old VOB database. Alternatively, you can use a standard *rm* command at a later time to discard it — for example, after you have mounted the reformatted VOB and verified its accessibility.
- By default, the ASCII dump files are created within the VOB storage directory. You can use the `-to` option to create these files in another location.

Following are *reformatvob*'s disk-space requirements, based on the size of the existing VOB database as the unit:

Data Structure	Space Required	Need for Space Eliminated by
existing VOB database	100%	`-rm`
ASCII dump files	100%	`-to`
elbow room	10%	(always required)

Thus, if you use neither `-rm` nor `-to`, *reformatvob*'s disk-space needs total approximately 210% of the size of the VOB database.

Restarting an Interrupted Reformat

There are no ill effects if a *reformatvob* command is interrupted (for example, by a system crash). Enter the *reformatvob* command again to complete the reformatting. If *reformatvob* is interrupted after the dump phase completes, a subsequent invocation automatically starts with the load phase.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, root user.
Locks: No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Partial Reformat. *Default:* Performs a complete reformat, including both the dump and load phases.

–dum·p Performs only the first phase of the reformatting process — creating an ASCII *dump* of the current VOB database.

–loa·d Performs only the second phase of the reformatting process — creating a new VOB database using a previously created ASCII dump.

Preserving a Backup of the VOB Database. *Default:* The original VOB database directory (subdirectory *db* of the VOB storage directory) is preserved through renaming. During the dump phase, it is renamed to *db.reformat*; during the load phase, it is renamed again, to a name that includes a date stamp (for example, *db.02.18*).

–rm Deletes the original VOB database during the load phase.

Confirmation Step. *Default:* Before beginning its work, *reformatvob* has you confirm that you wish to reformat the VOB database.

–for·ce Suppresses the confirmation step.

Alternate Location for ASCII Dump Files. *Default:* The dump phase creates the ASCII dump files within the VOB storage directory.

–to *dumpfile-dir-pname*
(do not use in conjunction with `–load`) Creates the ASCII dump files within the specified directory, which must not already exist.

VOB Registry Options. *Default:* Using the *vob-storage-dir-pname* argument, *reformatvob* creates or updates the *vob_object* registry file; it leaves the *vob_tag* registry file unchanged. The following options update the VOB-tag entry; see the *mkvob* manual page for more information.

–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*

Specifying the VOB. *Default:* None.

vob-storage-dir-pname
The pathname of a VOB storage directory.

See also the descriptions of `–host`, `–hpath`, and `–gpath` above.

EXAMPLES

- Reformat a VOB whose storage directory is */home/jones/tut/tut.vbs*.

  NOTE: This example shows selected status messages only; *reformatvob* actually produces much more verbose messages.

  ```
  % cd
  ```

```
% cleartool reformatvob /home/jones/tut/tut.vbs
Reformat versioned object base "/home/jones/tut/tut.vbs"?  [no] y
Dumping database...
Dumper done.
Dumped versioned object base "/home/jones/tut/tut.vbs".
Loading database...
Loader done.
Loaded versioned object base "/home/jones/tut/tut.vbs".
```

SEE ALSO

cleartool subcommands: lsvob, mkvob, mktag, mount, register
db_dumper, registry_ccase, vob, vob_scrubber

NAME register – create an entry in the vob_object or view_object registry file

SYNOPSIS

- Register a VOB:

  reg·ister –vob [**–rep·lace**]
  [**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
  vob-storage-dir-pname

- Register a view:

  reg·ister –view [**–rep·lace**]
  [**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
  view-storage-dir-pname

DESCRIPTION

Creates or replaces an entry in the *vob_object* registry file or the *view_object* registry file:

- The *vob_object* registry file, located at */usr/adm/atria/rgy/vob_object*, has an entry for each VOB in the local area network.
- The *view_object* registry file, located at */usr/adm/atria/rgy/view_object*, has an entry for each view in the local area network.

ClearCase software uses these registries to determine the physical storage locations of VOBs and views. In most commands, you reference a VOB or view with its *VOB-tag* or *view-tag*. These tags "point to" the physical storage directories. Note that *register* has no effect on the *vob_tag* or *view_tag* registry files.

Use *register* to update an existing registry entry, or to re-register a VOB or view that was temporarily removed from service with *unregister*.

Other Commands that Affect Storage Registries

The *mkview* and *mkvob* commands automatically add an entry to the appropriate registry; the *rmview* and *rmvob* commands remove registry entries. You can use the *unregister* command to remove an existing entry.

The *reformatvob* command updates a VOB's *vob_object* registry entry (or creates one, if necessary).

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

View/VOB Specification. *Default:* None.

–vob Registers a VOB storage directory.

–view Registers a view storage directory.

Overwriting an Existing Entry. *Default:* An error occurs if the view or VOB storage directory already has an entry in the registry.

–rep·lace Replaces an existing registry entry. (No error occurs if there is no pre-existing entry.)

Specifying the Location of the Storage Directory. *Default:* None. You must specify a VOB or view storage directory. In rare cases, you may need to specify host-local and global access path information as well. See the *mkvob* and *mkview* manual pages for descriptions of how to use these options and arguments to specify VOB and view storage directories.

view-storage-dir-pname
vob-storage-dir-pname
–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*

EXAMPLES

- Register a VOB storage directory that was previously unregistered with the command `unregister –vob /vobstore/vob2.vbs`

```
% cleartool register –vob /vobstore/vob2.vbs
```

- Register a view storage directory.

```
% cleartool register –view /viewstore/view3.vws
```

FILES

/usr/adm/atria/rgy/vob_object
/usr/adm/atria/rgy/view_object

SEE ALSO

cleartool subcommands: unregister, mkvob, mkview, mktag, mount, umount
registry_ccase

NAME reserve – convert an unreserved checkout to reserved

SYNOPSIS

res·erve [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *pname* …

DESCRIPTION

Changes the "checkout status" of a checked−out version of an element to *reserved*. An `reserve checkout of version` event record is written to the VOB database.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Elements. *Default:* None.

pname … One or more pathnames, each of which specifies an element. The checkout in the current view is changed, unless you use a view-extended pathname to specify another view.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `−nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Change the checkout status of an element to reserved.

```
% cleartool reserve util.c
Changed checkout to reserved for "util.c" branch "/main".
```

- Verify that you are the only user with a checkout of a certain file, then convert your checkout from unreserved to reserved.

```
% cleartool lscheckout util.c
14-Mar.13:48   drp       checkout version "util.c" from /main/3 (unreserved)
  "experiment with algorithm for returning time"
```

```
% cleartool reserve util.c
Changed checkout to reserved for "util.c" branch "/main".
```

SEE ALSO

cleartool subcommands: checkin, checkout, lscheckout, uncheckout, unreserve
profile_ccase

NAME rmattr – remove an attribute from an object

SYNOPSIS

- Remove attribute from a file system object:

  rmattr [**–ver·sion** *version-selector*] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  attribute-type-name pname …

- Remove attribute from a hyperlink:

  rmattr –hli·nk [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  attribute-type-name hlink-selector …

DESCRIPTION

Removes one or more *attributes* from VOB-database objects. Attributes can be attached to objects by the *mkattr* command and by triggers (`mktrtype -mkattr`). See the *mkattr* manual page for a list of objects to which attributes can be attached.

rmattr deletes an *instance* of an attribute type object. To delete the attribute type object itself, use the *rmtype* command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, attribute type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Object. *Default: rmattr* interprets command arguments as the names of file system objects.

–hli·nk Indicates that command argument(s) name hyperlink objects, not file system objects.

Specifying a File System Object. *Default:* None.

pname … One or more pathnames, indicating objects from which attributes are to be removed. If you don't use the `–version` option, then:

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A VOB-extended pathname specifies an element, branch, or version, independent of view.

See the *mkattr* manual page for examples of *pname* arguments.

–ver·sion *version-selector*
Specifies the version from which the attribute is to be removed. See the *version_selector* manual page for syntax details.

Specifying a Hyperlink Object. *Default:* None.

hlink-selector

A hyperlink-selector argument takes this form:

hyperlink-type-name@hyperlink-ID[*@pname-in-vob*]

Hyperlinks are not file system objects — you cannot specify them with shell wildcards. The final component is required only for a hyperlink in another VOB. Examples:

```
DesignFor@598f
RelatesTo@58843@/vobs/monet
```

Specifying the Attribute to be Removed. *Default:* None.

attribute-type-name

An existing attribute type.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**

Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Remove the *Confidence_Level* attribute from the version of *msg.c* in the view.

```
% cleartool rmattr Confidence_Level msg.c
Removed attribute "Confidence_Level" from "msg.c@@/main/1".
```

- Remove the attribute *TESTED* from the most recent version of *hello.h* on the *main* branch that has the attribute value "FALSE".

```
% cleartool rmattr –version '/main/{TESTED=="FALSE"}' TESTED hello.h
Removed attribute "TESTED" from "hello.h@@/main/2".
```

- Remove the *Responsible* attribute from the *main* branch of *hello.c*.

```
% cleartool rmattr Responsible hello.c@@/main
Removed attribute "Responsible" from "hello.c@@/main".
```

- Remove the *Author* attribute from a hyperlink of type *DesignDoc*.

```
cleartool rmattr -hlink Author DesignDoc@393@/usr/hw
Removed attribute "Author" from "DesignDoc@393@/usr/hw".
```

SEE ALSO

cleartool subcommands: lstype, mkattr, mkattype, rmtype, rntype
events_ccase, profile_ccase, version_selector

NAME rmbranch – remove a branch from the version tree of an element

SYNOPSIS

rmbranch [**–for·ce**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* …

DESCRIPTION

This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases.

rmbranch deletes one or more branches from their elements. For each branch, this entails:

- removal from the entire branch structure from the VOB database: branch object and version objects
- removal of all *meta-data* items (labels, attributes, hyperlinks, and triggers) that were attached to the deleted objects
- removal of all event records for the deleted objects
- (file elements only) removal of the data container(s) that hold the deleted versions' file system data

  NOTE: If all of an element's versions are stored in a single data container, the deleted versions are removed *logically*, not physically.

A `destroy sub-branch` event record is created for the parent branch of the deleted branch.

Restrictions: You cannot delete a branch that is checked out. You cannot delete an element's *main* branch.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: branch creator, element owner, VOB owner, or root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, pool (non-directory elements only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Confirmation Step. *Default: rmbranch* prompts for confirmation before deleting anything.

–for·ce Suppresses the confirmation step.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Branches to be Removed. *Default:* None.

pname … One or more VOB-extended pathnames, indicating the branch(es) to be deleted. Examples:

```
foo.c@@/main/bugfix
/vobs/proj/include/proj.h@@/main/temp_482
```

EXAMPLES

- Delete the *maintenance* branch of element *util.c*.

```
% cleartool rmbranch util.c@@/main/maintenance
Branch "util.c@@/main/maintenance" has 0 sub-branches, 2 sub-versions
Remove branch, all its sub-branches and sub-versions?  [no] yes
Removed branch "util.c@@/main/maintenance".
```

- Verify, with the *lsvtree* command, that element *msg.c* has a *patch2* branch. Then, delete that branch without prompting for confirmation.

```
% cleartool lsvtree –branch /main/patch2 msg.c
msg.c@@/main/patch2
msg.c@@/main/patch2/1
```

```
% cleartool rmbranch –force msg.c@@/main/patch2
Removed branch "msg.c@@/main/patch2".
```

SEE ALSO

cleartool subcommands: lsvtree, mkbranch, mkbrtype, rmver
profile_ccase

NAME rmdo – remove a derived object from a VOB

SYNOPSIS

- Remove individual derived objects:

  rmdo *do-pname …*

- Remove collections of derived objects:

  rmdo { **–a·ll** | **–zer·o** } [*pname …*]

DESCRIPTION

Deletes one or more *derived objects* (DOs) from a VOB. The details of this process depend on whether or not the derived object is *shared* (as indicated by the `lsdo –long` command). The overall principle is that *rmdo* affects VOB storage only — it does not affect view storage:

- For a *shared derived object*, whose *data container* is in VOB storage, *rmdo* deletes the entry in the VOB database, and also deletes the data container file (from one of the VOB's derived object storage pools).
- For an *unshared derived object*, whose data container is in view-private storage, *rmdo* deletes the entry from the VOB database, but does *not* delete the data container from view storage. The data container is an ordinary file, which can still be listed, executed, and so on; but it cannot be a candidate for configuration lookup. The `ls -long` command lists it with a `[no config record]` annotation. To delete the data file, use the UNIX *rm*(1) command.

In either case, *rmdo* also deletes the associated configuration record if it is no longer needed — that is, if both these conditions hold:

- no other *sibling* derived object (created in the same build script execution) still exists
- the derived object was not a build dependency (subtarget) of another derived object that still exists

Command options enable deletion of groups of derived objects:

- derived objects created at the same UNIX pathname, across all views
- derived objects that are no longer referenced in any view (that is, whose *reference counts* are zero)
- all derived objects (or all zero-referenced derived objects) in a directory)

rmdo does not delete *DO versions* — derived objects that have been checked in as versions of elements. Use *rmver* for this purpose.

CAUTION

Deleting a shared derived object causes the next UNIX-level access to return an `I/O error` status. This may, in turn, cause *clearmake* to log an `INTERNAL`ERROR message to the ClearCase *error_log* file. The derived object's name is removed from the directory by this UNIX-level access; thus, subsequent accesses return `not found` errors. To avoid such situations, use the `–zero` option when you enter an *rmdo* command.

SCRUBBING OF DERIVED OBJECTS

ClearCase includes a utility, *scrubber*, that deletes derived objects — both the entries in the VOB database and (for shared derived objects) the data containers in the VOB's storage pools. By default, a *crontab*(1) script runs *scrubber* once each day.

Each derived object pool has scrubbing parameters, which you can modify with the `mkpool -update` command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, root user.
Locks: An error occurs if any of the following objects are locked: VOB, pool. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of Like-Named Derived objects. *Default:* Deletes at most one DO for each file name specified with command arguments. A file name with a *DO-ID* (for example, `hello.o@@24-Mar.11:32.412`), specifies exactly which DO to delete. A standard or view-extended pathname specifies the DO that appears in the view.

To determine the DO-IDs of derived objects, use *lsdo*.

–a·ll Deletes all derived object(s) at a given pathname, no matter what view they were created in, or currently appear in.

CAUTION: This can disrupt work taking place in other developers' views.

–zer·o Similar to `–all`, but deletes only those derived objects with zero reference counts.

Specifying Derived Objects. *Default:* If you use `–all` or `–zero`, the default pathname is ".", which specifies all DO file names in the current working directory. If you use neither of these options, there is no default — you must supply at least one argument.

do-pname …

Pathnames of one or more individual derived objects. A name with a DO-ID, such as *foo@@10-Nov.10:14.27672*, specifies a particular derived object, irrespective of view. A standard UNIX pathname or view-extended pathname specifies the derived object that appears in a view.

pname … (use with `–all` or `–zero`) One or more standard or view-extended pathnames, each of which can name a file or directory:

- A file name specifies a collection of DOs built at the same pathname.
- A directory name is equivalent to a list of all the file names of DOs built in that directory — even file names that do not currently appear in the view (perhaps after a `make clean`).

EXAMPLES

- Delete the derived object *hello.o@@24-Mar.11:32.412*.

```
% cleartool rmdo hello.o@@24–Mar.11:32.412
Removed derived object "hello.o@@24-Mar.11:32.412".
```

- Delete all derived objects named *hello* in the current working directory.

```
% cleartool rmdo –all hello
Removed derived object "hello@@23-Mar.14:16.178".
Removed derived object "hello@@23-Mar.19:25.394".
```

- Delete all zero-referenced derived objects in the *hworld* directory.

```
% cleartool rmdo –zero hworld
Removed derived object "hworld/hello.o@@23-Mar.20:42.373".
Removed derived object "hworld/hello.o@@23-Mar.20:36.228".
Removed derived object "hworld/hello@@23-Mar.20:42.382".
Removed derived object "hworld/hello@@23-Mar.20:36.234".
Removed derived object "hworld/util.o@@23-Mar.20:42.376".
Removed derived object "hworld/util.o@@23-Mar.20:36.231".
```

SEE ALSO

cleartool subcommands: catcr, diffcr, ls, lsdo, mkpool, rmver
clearaudit, crontab(1), rm(1), scrubber, crontab_ccase

NAME rmelem – remove an element from a VOB

SYNOPSIS

rmelem [**–for·ce**] [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* …

DESCRIPTION

This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases. In many cases, it is better to use the 'rmname' command.

rmelem completely deletes one or more elements. For each element, this entails:

- removal of the entire version tree structure from the VOB database: element object, branch objects, and version objects
- removal of all *meta-data* items (labels, attributes, hyperlinks, and triggers) that were attached to the deleted objects
- removal of all event records for the deleted objects
- (file elements only) removal of the data container(s) that hold the element's file system data from its source storage pool
- removal of all references to the element from versions of the VOB's directory elements. (This means that subsequent listings and comparisons of those directory versions will be "historically inaccurate".)

A `destroy element` event record is created for the element's VOB.

RESTRICTION: you cannot delete an element if any of its versions are checked out. (But it is not necessary to *checkout* the parent directory before deleting one of its subdirectories.)

Deleting a Directory Element

Deleting a directory element may cause some other elements to be *orphaned*: no longer cataloged in any version of any directory. *rmelem* displays a message and moves an orphaned element to the VOB's *lost+found* directory:

```
cleartool: Warning: Object "foo.c" no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory
as "foo.c.a0650992e2b911ccb4bc08006906af65".
```

(See the *mkvob* manual page for a description of this directory.)

Each derived object in the deleted directory is also moved to *lost+found*. The derived object has no data, but you can use it in such commands as *lsdo* and *catcr*.

View-private objects in the deleted directory are temporarily *stranded*, but can be transferred to the view's own *lost+found* directory:

1. Use *lsprivate* to locate stranded files and to determine the ClearCase-internal identifier of the deleted directory element:

```
% cleartool lsprivate –vob /tmp/david_phobos_hw
  ⋮
#<Unavailable-VOB-1>/<DIR-c8051152.e2ba11cc.b4c0.08:00:69:06:af:65>/myfile
```

2. Use *recoverview* to move all the stranded file(s) for the deleted directory:

```
% cleartool recoverview –dir c8051152.e2ba11cc.b4c0.08:00:69:06:af:65 –tag myview
Moved file /usr/people/david/myview.vws/.s/lost+found/5ECC880E.00A5.myfile
```

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, pool (non-directory elements only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Confirmation Step. *Default: rmelem* prompts for confirmation before deleting anything.

–for·ce Suppresses the confirmation step.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Elements to be Removed. *Default:* None.

pname … One or more pathnames, indicating the element(s) to be deleted. An extended pathname to a particular version or branch of an element references the element itself.

EXAMPLES

- Delete the file element *convolution.c*.

```
% cleartool rmelem convolution.c
Element "convolution.c" has 1 branches, 2 versions, and is entered
in 6 directory versions.
Remove element, all its branches and versions and modify all directory
versions containing element?  [no] yes
Removed element "convolution.c".
```

- Delete the directory element *release*. Note that an orphaned element, *hello*, is moved to the VOB's *lost+found* directory.

```
% cleartool rmelem release
Element "release" has 1 branches, 9 versions, and is entered
in 35 directory versions.
Remove element, all its branches and versions and modify all directory
versions containing element?  [no] yes
cleartool: Warning: Object "hello" no longer referenced.
Object moved to vob lost+found directory as
        "hello.5d400002090711cba06a080069061935".
Removed element "release".
```

SEE ALSO

cleartool subcommands: mkelem, mkvob, rmbranch, rmname, rmver
profile_ccase

NAME rmhlink – remove a hyperlink object

SYNOPSIS

rmhlink [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *hlink-selector …*

DESCRIPTION

Removes one or more *hyperlinks* from VOB-database objects. Hyperlinks can be attached to objects by the *mkhlink* command and by triggers (`mktrtype -mkhlink`). See the *mkhlink* manual page for a list of objects to which hyperlinks can be attached.

rmhlink deletes a *reference* to a hyperlink type object. To delete the hyperlink type object itself, use the *rmtype* command.

To list existing hyperlinks, use the *describe* command, or use the *find* command with the `hltype` primitive.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Hyperlink(s) to be Removed. *Default:* None.

hlink-selector …
One or more names of hyperlink objects, in this form:

hyperlink-type-name@hyperlink-ID[*@pname-in-vob*]

Hyperlinks are not file system objects — you cannot specify them with shell wildcards. The final component is required only for a hyperlink in another VOB. Examples:

```
DesignFor@598f
RelatesTo@58843@/vobs/monet
```

EXAMPLES

- Remove a hyperlink of type *tested_by* from the element *cm_add.c*. Use *describe* to determine the hyperlink selector.

```
% cleartool describe cm_add.c@@
file element "cm_add.c@@"
  created 08-Dec-92.12:12:52 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  element type: c_source
  source pool: sdft  cleartext pool: cltxt2
  Hyperlinks:
```

```
    tested_by@714@/usr/hw  /usr/hw/src/cm_add.c@@
        "edge effects" -> /usr/hw/src/edge.sh@@ "regression A"
```

```
% cleartool rmhlink tested_by@714
Removed hyperlink "tested_by@714".
```

- Remove two hyperlinks from the *src* directory. Use *describe* to determine the hyperlink selectors.

```
% cleartool describe src
directory version "src@@/main/9"
  created 08-Dec-92.12:23:46 by Chuck Jackson (test user) (jackson.dvt@oxygen)
  element type: directory
  Hyperlinks:
    h3@1320@/usr/hw  /usr/hw/src@@/main/9 ->
    h1@1324@/usr/hw  /usr/hw/src/hello@@/main/1 -> /usr/hw/src@@/main/9
    h2@1329@/usr/hw  /usr/hw/bin@@/main/1 -> /usr/hw/src@@/main/9
```

```
% cleartool rmhlink h1@1324 h2@1329
Removed hyperlink "h1@1324".
Removed hyperlink "h2@1329".
```

SEE ALSO

cleartool subcommands: describe, lshistory, mkhlink, mkhltype, rmtype
xclearcase, profile_ccase

NAME rmlabel – remove a version label from a version

SYNOPSIS

rmlabel [**–ver·sion** *version-selector*] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
label-type-name pname …

DESCRIPTION

Removes one or more *version labels* from versions of elements. Labels can be attached to versions by the *mklabel* command and by triggers (`mktrtype -mklabel`).

rmlabel deletes a *reference* to a label type object. To delete the label type object itself, use the *rmtype* command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, label type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Versions to be Labeled. *Default:* None.

pname … One or more pathnames, indicating versions from which the label is to be removed. What kind of pathname is valid depends on how the label has been used:

– If the label has been used only once in an element's version tree, you can specify the element itself, or any of its branches or versions:

```
foo.c                          (version selected by view)
foo.c@@                        (element itself)
foo.c@@/main/rel2_bugfix       (branch of element)
```

– If the label has been used multiple times, you must specify either the version to which the label is attached, or the branch on which that version resides.

```
foo.c                          (version selected by view)
foo.c@@/REL1                   (version specified by label)
foo.c@@/main/rel2_bugfix/3     (version specified by version–ID)
foo.c@@/main/rel2_bugfix       (branch on which version resides)
```

Using the `–version` option modifies the way in which this argument is interpreted.

–ver·sion *version-selector*

Specifies the version from which the label is to be removed. See the *version_selector* manual page for syntax details. Using this option overrides a version-extended pathname. For example:

```
% cleartool rmlabel XXX util.c@@/REL1                 (removes label from version 'REL1')
% cleartool rmlabel -ver /main/3 XXX util.c@@/REL1    (removes label from version '/main/3')
```

Specifying the Label to be Removed. *Default:* None.

label-type-name
An existing label type.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Remove the label *REL3* from a version of *msg.c* without specifying which version. (Assumes the label is attached to one version only.)

```
% cleartool rmlabel REL3 msg.c
Removed label "REL3" from "msg.c" version "/main/1".
```

- Remove the label *REL2* from the version of element *util.c* specified by a version selector.

```
% cleartool rmlabel –version /main/REL2 REL2 util.c
Removed label "REL2" from "util.c" version "/main/1".
```

- Remove the label *REL1.1* from version 1 on the *maintenance* branch of file element *util.c*. Use a version-extended pathname to indicate the version.

```
% cleartool rmlabel REL1.1 util.c@@/main/maintenance/1
Removed label "REL1.1" from "util.c" version "/main/maintenance/1".
```

SEE ALSO

cleartool subcommands: lstype, mklbtype, rmtype, rntype
profile_ccase, version_selector

NAME rmmerge – remove a merge arrow from an element´s version tree

SYNOPSIS

rmmerge [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *from-pname to-pname*

DESCRIPTION

Deletes an existing *merge arrow* (a hyperlink of the predefined type *Merge*) between two versions of an element. Thus, this command is a specialized form of the *rmhlink* command. The two commands have an identical result; they differ only in the way you specify the merge arrow:

- With *rmhlink*, you specify the merge arrow itself, using a hyperlink-selector.
- With *rmmerge*, you specify the versions linked by the merge arrow.

To list existing merge arrows, use the *describe* command, or use the *find* command with the `hltype` primitive. For example:

```
% cleartool describe util.c
version "util.c@@/main/3"
  created 05-Apr-92.17:01:12 by Allison (akp.user@starfield)
  element type: text_file
  Hyperlinks:
    Merge@148@/usr/tmp/poolwk
        /usr/tmp/poolwk/src/util.c@@/main/rel2_bugfix/1 ->
                /usr/tmp/poolwk/src/util.c
```

Renaming the 'Merge' Hyperlink Type

Renaming the predefined hyperlink type for merge arrows does not defeat *rmmerge*. You just specify the element's versions — *rmmerge* automatically determines the hyperlink type used for merge arrows in that element's VOB.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Versions. *Default:* None.

from-pname, to-pname
Extended pathnames of the versions connected by the merge arrow. The order in which you specify the versions is important: the source version first, the target version second.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLE

- Remove the merge arrow between the latest version on the *rel2_bugfix* branch and the version of *util.c* in the view.

```
% cleartool rmmerge  util.c@@/main/rel2_bugfix/LATEST util.c
Removed merge from "util.c@@/main/rel2_bugfix/1" to "util.c".
```

SEE ALSO

cleartool subcommands: describe, merge, rmhlink, rntype, xmerge
profile_ccase

NAME rmname – remove the name of an element or VOB symbolic link from a directory version

SYNOPSIS

rm·name [**−c** *comment* | **−cq** | **−cqe** | **−nc**] *pname* …

DESCRIPTION

NOTE: A name can be removed from a directory only if that directory is checked-out. *rmname* automatically appends an appropriate line to the directory's checkout comment.

rmname is analogous to the UNIX *unlink*(2) system call: it modifies one or more checked-out directories by removing the names of elements and/or VOB symbolic links. Old versions of the directories remain unaffected — the names continue to be cataloged in the old versions.

Example: Suppose you checked out version 3 of a directory named *a.dir*. Only your view sees this directory version while it is checked out. The command `rmname foo.c` deletes the name "foo.c" from the checked-out version of the directory, but leaves references to *foo.c* in earlier versions (if any) intact. When you checkin the directory, all views will be able to access the new version 4, which does not include *foo.c*.

Keep the following points in mind:

- *rmname* does not delete elements themselves, only references to elements. Use *rmelem* (very carefully) to delete elements from their VOBs.
- Removing the last reference to an element name causes the element to be *orphaned*. Such elements are automatically moved to the VOB's *lost+found* directory. (See the *mkvob* command for details.)
- Removing the last reference to a VOB symbolic link deletes the link object. (VOB symbolic links do not migrate to the VOB's *lost+found* directory.)

Undoing the 'rmname' Command

To restore a directory entry for an element that has been removed with *rmname*, use the *ln* command to create a *VOB hard link* to the element's entry in any previous version of the directory. For example:

```
cleartool checkout src                                  (checkout parent directory)
cleartool rmname src/msg.c                              (oops!)
cleartool ln src@@/main/LATEST/msg.c src/msg.c          (restore deleted name)
```

If there is no such "old" entry, then the element is *orphaned*; ClearCase will have moved it to its VOB's *lost+found* directory. You can move/rename the element to its proper location with the ClearCase *mv* command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: (directory must be checked out; see *checkout* permissions). *Locks:* An error occurs if any of the following objects are locked: VOB. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Names to be Removed. *Default:* None.

pname … One or more pathnames, specifying the elements and/or VOB symbolic links whose names are to be removed from their parent directory. You can specify an element itself, or any of its branches or versions.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

NOTE: Examples assume that the current working directory is checked out.

- Delete the name *util.c* from the current directory version.

```
% cleartool rmname util.c
Removed "util.c".
```

- Delete the last reference to the directory element *subd* from the current directory version.

```
% cleartool rmname subd
cleartool: Warning: Object "subd" no longer referenced.
Object moved to vob lost+found as
        "subd.5a200007ed11f0d709066505efe922a8".
Removed "subd".
```

SEE ALSO

cleartool subcommands: ln, mkvob, mv, rmelem, rmver
profile_ccase, unlink(2)

NAME rmpool – remove a VOB storage pool

SYNOPSIS

rmpool [**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–vob** *pname-in-vob*] *pool-name* …

DESCRIPTION

Deletes one or more *storage pool* directories from a VOB, along with all the *data container* files stored within them.

Restrictions

Before removing a storage pool, you must reassign all its currently-assigned elements to a different pool, using the *chpool* command. Otherwise, *rmpool* aborts with an `elements using pool` error. To list all the elements in a source or cleartext pool, use a *find* command. For example:

```
% cleartool find –all –element 'pool(source_2)' –print
```

This command does not work with pools created with the obsolete `mkpool -mount` command.

This command does not work with derived object pools.

Deleting Derived Object Pools

There is no way to move a shared derived object from one pool to another. Thus, you can delete a derived object pool only if:

- no directory elements have been assigned to the pool, or …
- all data containers in the pool have been removed by the *scrubber* program or *rmdo* commands, and each directory element that currently uses the pool has been assigned to a different derived object pool

OPTIONS AND ARGUMENTS

Specifying the VOB. *Default:* Removes a pool from the VOB containing the current working directory.

–vob *pname-in-vob*

The VOB whose pool is to be removed. *pname-in-vob* can be the pathname of any object within the VOB.

Specifying the Pools to be Removed. *Default:* None.

pool-name …

One or more names of existing storage pools. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: pool creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, pool. See the "Permissions Checking" section of the *cleartool* manual page.

EXAMPLES

Change all elements using the *c_source_pool* to use the default source pool (*sdft*) instead. Then, delete *c_source_pool*.

```
% cleartool find . −all −element 'pool(c_source_pool)' \
      −exec 'cleartool chpool −force sdft $CLEARCASE_PN'
Changed pool for "/usr/hw/src" to "sdft".
Changed pool for "/usr/hw/src/libutil.a" to "sdft".
  :
  :
% cleartool rmpool c_source_pool
Removed pool "c_source_pool".
```

SEE ALSO

cleartool subcommands: describe, chpool, find, lspool, mkpool, rmdo, rnpool
profile_ccase, scrubber, chmod(1)

NAME rmtag – remove a view-tag or a VOB-tag from the network-wide storage registry

SYNOPSIS

rmtag { **–vie·w** | **–vob** } [**–reg·ion** *network-region* | **–all**] *view-or-vob-tag …*

DESCRIPTION

Removes one or more entries from the network's *view_tag* registry file or *vob_tag* registry file. See the *registry_ccase* manual page for a discussion of registry files.

Restrictions

You cannot remove a tag that is currently in use:

- A VOB-tag is in use if the VOB is active on any host in the network region. Use the *cleartool* subcommand *umount* to deactivate a VOB on all hosts in the region before removing its tag.
- A view-tag is in use if any user process is set to the view specified by this tag, or if any user process has a current working directory that is a view-extended pathname based on this tag.

A VOB or view must always have a tag in its "home region" — the network region of the host where the VOB or view storage directory physically resides. If you remove a home-region tag, create a new one immediately.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Tag. *Default:* None.

–vie·w Removes one or more view-tags.

–vob Removes one or more VOB-tags.

Specifying a Network Region. *Default:* Removes tags that are defined for the local host's network region. (A host's network region is listed in file */usr/adm/atria/rgy/rgy_region.conf*.) See the *registry_ccase* manual page for a discussion of *network regions*.

–reg·ion *network-region*
Removes a tag defined for the specified *network region*. An error occurs if the region does not already exist.

–all Removes a tag from all network regions for which it is defined.

Specifying the Tag(s). *Default:* None.

view-tag … One or more view-tags to be removed.

vob-tag … One or more VOB-tags to be removed.

EXAMPLES

- Remove the view-tag *R2alpha* from the view registry.

```
% cleartool rmtag –view R2alpha
%
```

SEE ALSO

cleartool subcommands: mktag, mkview, mkvob, rmview, rmvob
registry_ccase

NAME rmtrigger – remove trigger from an element

SYNOPSIS

rmtrigger [**−r·ecurse**] [**−nin·herit** | **−nat·tach**] [**−c** *comment* | **−cq** | **−cqe** | **−nc**]
trigger-type-name pname …

DESCRIPTION

Removes an attached trigger from one or more elements. By default, *rmtrigger* removes the trigger from both the *attached list* and the *inheritance list* (if a directory element). You can modify the default action for directory elements with the `−ninherit` and `−nattach` options.

The specified *trigger-type-name* is unaffected by *rmtrigger*. To delete the trigger type, use the *rmtype* command. Note that you can remove an attached trigger from an element even if the trigger type is obsolete.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, trigger type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Removing Triggers from an Entire Subdirectory Tree. *Default:* If a *pname* argument names a directory element, the trigger is removed only from the element itself, not from any of the existing elements within it.

−r·ecurse Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are *not* traversed during the recursive descent into the subtree.

Manipulating the Trigger Lists of a Directory Element. *Default:* The trigger is removed from both of a directory element's trigger lists: its *attached list* and its *inheritance list*. Be careful when using the following options — they involve a "double negative".

−nin·herit (directory element only) The trigger is removed from the directory's attached list, but remains on its inheritance list. The trigger will not fire when the monitored operation is performed on the directory itself, but new elements created in that directory will inherit the trigger.

−nat·tach (directory element only) The trigger is removed from the directory's inheritance list, but remains on its attached list. The trigger will continue to fire when the monitored operation is performed on the directory itself, but new elements created in that directory will not inherit the trigger.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `−nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

−c *comment* , **−cq** , **−cqe** , **−nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Trigger Type. *Default:* None.

trigger-type-name
The name of an existing *element* trigger type.

Specifying the Elements. *Default:* None.

pname … One or more pathnames, specifying elements from which triggers (instances of the specified trigger type) are to be removed.

EXAMPLES

- Remove an attached trigger from *hello.c*.

```
% cleartool rmtrigger trig1 hello.c
Removed trigger "trig1" from attached list of "hello.c".
```

- Remove an attached trigger from the *hworld* directory's attached list, but leave it in the inheritance list.

```
% cleartool rmtrigger –ninherit trig1 src
Removed trigger "trig1" from attached list of "src".
```

- Remove an attached trigger from the *release* directory's inheritance list, but leave it in the attached list.

```
% cleartool rmtrigger –nattach trig1 release
Removed trigger "trig1" from inheritance list of "release".
```

SEE ALSO

cleartool subcommands: describe, lock, lstype, mktrigger, mktrtype, rmtype, unlock
profile_ccase

NAME rmtype – remove a type object from a VOB

SYNOPSIS

rmtype { **–elt·ype** | **–brt·ype** | **–lbt·ype** | **–att·ype** | **–hlt·ype** | **–rpt·ype** | **–trt·ype** [**–ign·ore**] }
[**–rma·ll** [**–for·ce**]] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
[**–vob** *pname-in-vob*] *type-name* …

DESCRIPTION

Removes one or more *type objects* from a VOB.

RESTRICTION: You cannot delete a type object if there are any instances of that type. For example, if any version of any element is labeled *REL1*, you cannot delete the *REL1* label type (unless you specify the `–rmall` option).

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Type Object. *Default:* None.

–elt·ype (element type)
–brt·ype (branch type)
–lbt·ype (label type)
–att·ype (attribute type)
–hlt·ype (hyperlink type)
–trt·ype (trigger type)
–rpt·ype (replica type)
(mutually exclusive — exactly one required) Specifies the kind of type object to be deleted.

–ign·ore (for use with '–trtype' only)
Removes a trigger type even if a previously defined pre-operation trigger would otherwise prevent it from being removed.

Removing Instances of the Type. *Default:* If there are any instances of a specified type object, *rmtype* refuses to remove the type object.

–rma·ll Removes all instances of a type, and then proceeds to remove the type object itself.

CAUTION: This option potentially destroys a great deal of data.

–for·ce (for use with '–rmall' only)
By default, *rmtype* prompts for confirmation when you use the `–rmall` option to request removal of all instances of a type. The `–force` option suppresses the confirmation step.

Specifying the VOB. *Default:* Removes types from the VOB that contains the current working directory.

–vob *pname-in-vob*
The VOB whose type(s) are to be removed. *pname-in-vob* can be any location within the VOB.

Specifying the Type Objects to be Removed. *Default:* None.

type-name …
One or more names of existing type objects, of the specified kind.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Delete the branch type *patch3*.

```
% cleartool rmtype –brtype patch3
Removed branch type "patch3".
```

- Delete the attribute type *QA_date*.

```
% cleartool rmtype –attype QA_date
Removed attribute type "QA_date".
```

- Delete all branches of type *expmnt3* (along with all the versions on those branches and any subbranches); then delete the *expmnt3* branch type itself:

```
% cleartool rmtype –rmall –brtype expmnt3
There are 1 branches of type "expmnt3".
Remove branches (including all sub-branches and sub-versions)?  [no] yes
Removed branches of type "expmnt3".
Removed branch type "expmnt3".
```

- Delete the hyperlink type *design_doc*.

```
% cleartool rmtype –hltype design_doc
Removed hyperlink type "design_doc".
```

- Remove all instances of the label type *REL2*, then delete the label type.

```
% cleartool rmtype –lbtype –rmall REL2
There are 7 labels of type "REL2".
Remove labels?  [no] yes
Removed labels of type "REL2".
Removed label type "REL2".
```

- Delete the trigger type *trig1*. Use the –ignore option to ensure that the command executes without interference from a previously defined trigger.

```
% cleartool rmtype –trtype –ignore trig1
Removed trigger type "trig1".
```

SEE ALSO

cleartool subcommands: describe, lshistory, lstype, mkeltype, mklbtype, mkbrtype, mkattype, mktrtype, mkhltype, rntype
profile_ccase

NAME rmver – remove a version from the version tree of an element

SYNOPSIS

rmver [**–for·ce**] [**–xbr·anch**] [**–xla·bel**] [**–xat·tr**] [**–xhl·ink**] [**–dat·a**]
[**–ver·sion** *version-selector* | **–vra·nge** *low-version high-version*]
[**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname …*

DESCRIPTION

This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases.

rmver deletes one or more versions from their elements. For each version, this entails:

- removal of the version object from the VOB database
- removal of all *meta-data* items (labels, attributes, hyperlinks, and triggers) that were attached to the deleted version
- removal of all event records for the deleted version
- (file elements only) removal of the data container(s) that hold the deleted version's file system data

  NOTE: If an element's versions are all stored in a single data container, the deleted version is removed *logically*, not physically, and no disk space is freed.

A `destroy version` event record is created for the element.

Restrictions

You cannot delete a version from which someone currently has a checkout. You cannot delete version 0 on a branch, except by deleting the entire branch. (See *rmbranch*.)

Deleted Version-IDs

ClearCase never reuses the version-ID of a deleted version. There is no way to "collapse" a branch to fill the holes left by deleted versions. If a deleted version was the last version on a branch (say, version 6), the next *checkin* on that branch will create version 7.

A reference to a deleted version produces a `not found` or `no such file or directory` error.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: version creator, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, pool (non-directory elements only). See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Confirmation Step. *Default: rmver* prompts for confirmation before deleting anything.

–for·ce Suppresses the confirmation step.

Deleting 'Interesting' Versions. *Default: rmver* refuses to delete a version to which a version label, attribute, or hyperlink is attached, or at which a branch begins.

–xbr·anch Deletes a version even if one or more branches begin there. In the process, those branches (including all their versions and subbranches) are also deleted.

–xla·bel Deletes a version even if it has one or more version labels.

–xat·tr Deletes a version even if it has one or more attributes.

–xhl·ink Deletes a version even if it has one or more hyperlinks. This also destroys the hyperlink object, thus modifying the other object to which the hyperlink was attached.

CAUTION: Using this option can delete *merge arrows* (hyperlinks of type *Merge*) created by the *merge* command. This may destroy essential meta-data.

Data-Only Deletion. *Default: rmver* deletes both (1) the version object in the VOB database along with associated meta-data, and (2) the corresponding data container in a source storage pool.

–dat·a Deletes only the data for the specified version, leaving the version object, its subbranches, and its associated meta-data intact. In particular, this option preserves event records and enables continued access to the configuration record of a DO version.

CAUTION: Using this option implicitly invokes the `–xbranch`, `–xlabel`, `–xattr`, and `–xhlink` options, as well. That is, the data container will be deleted even if the version is "interesting".

Specifying the Versions to be Removed. *Default:* None.

pname … (required) One or more pathnames, indicating versions to be removed:

- A standard or view-extended pathname to an element specifies the version selected by the view.
- A version-extended pathname specifies a version, independent of view.

Use `–version` or `–vrange` to override these interpretations of *pname*.

–ver·sion *version-selector*
For each *pname*, removes the version specified by *version-selector*. This option overrides both version-selection by the view and version-extended naming. See the *version_selector* manual page for syntax details.

–vra·nge *low-version high-version*
For each *pname*, removes all versions between (but not including) the two specified versions. *low-version* and *high-version* must be on the same branch, and are specified in the same way as *version-selector*.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: `–nc`). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Delete the version of *msg.c* in the view.

```
% cleartool rmver msg.c
Removing these versions of "msg.c":
  /main/1
Remove versions?  [no] yes
Removed versions of "msg.c".
```

- Delete version 1 on the *rel2_bugfix* branch of element *util.c*, using a version selector to specify the version, suppressing confirmation prompting.

```
% cleartool rmver –force –version /main/rel2_bugfix/1 util.c
Removing these versions of "util.c":
  /main/rel2_bugfix/1
Removed versions of "util.c".
```

- Delete version 3 on the *main* branch of element *Makefile*, even if it has labels and/or attributes. Use a version-extended pathname to specify the version.

```
% cleartool rmver –xlabel –xattr Makefile@@/main/3
Removing these versions of "Makefile":
  /main/3 (labels, attributes)
Remove versions?  [no] yes
Removed versions of "Makefile".
```

- Delete all versions between 0 and LATEST on the *main* branch of element *hello.c*.

```
% cleartool rmver –vrange /main/0 /main/LATEST hello.c
Removing these versions of "hello.c":
  /main/1
  /main/2
Remove versions?  [no] yes
Removed versions of "hello.c".
```

- Delete version 2 on the *main* branch of *util.c*, even if there are one or more subbranches off that version. (The subbranches, if any, are also deleted.)

```
% cleartool rmver –xbranch util.c@@/main/2
Removing these versions of "util.c":
  /main/2 (subbranches)
Remove versions?  [no] yes
Removed versions of "util.c".
```

SEE ALSO

cleartool subcommands: describe, lshistory, lsvtree, rmbranch, rmelem, rmname
profile_ccase, version_selector

NAME rmview – remove a view storage directory / remove view-related records from a VOB

SYNOPSIS

- Remove view storage directory tree:

  rmview [**–for·ce**] { **–tag** *view-tag* | *view-storage-dir-pname* }

- Remove view-related records from a VOB:

  rmview [**–for·ce**] [**–vob** *pname-in-vob*] **–uuid** *view-uuid*

DESCRIPTION

The two forms of this command perform different, but related, tasks:

- removing a view storage directory
- purging view-related records from a VOB

Deleting a View-Storage Directory

If you specify a view by naming its view storage directory, *rmview* removes the entire view storage area. It also purges checkout records and derived object records relating to that view from all accessible VOBs. If the view is currently active, its associated *view_server* process is killed and its entry in the *viewroot* directory is removed.

By default, *rmview* refuses to delete a view storage area if any element is checked out to that view. You can override this behavior with the `–force` option.

rmview does not allow you to remove your own view (*set view* or *working directory view*). Be sure that the current working directory is not within the view storage area that you are deleting.

If the view was created with `mkview -ln`, its view-private objects are stored in a directory tree in an alternate location. *rmview* attempts to delete this directory tree; if it does not succeed, an error occurs and the view storage area remains unaffected.

Purging View-Related Records from a VOB

If you specify a view by its UUID (*universal unique identifier* — see below), *rmview* removes all checkout records and derived object records relating to that view from the VOB containing the current working directory. Use this form of the command when:

- Complete purging of view-related records from VOBs is not possible. (For example, some of the VOBs may be off-line at the time you remove the view.)
- A view storage area cannot be deleted with *rmview*, because it has become unavailable for another reason: disk crash, accidental deletion with UNIX *rm*(1), and so on.

You may need to use this form of *rmview* repeatedly, to delete from multiple VOBs all records relating to a view that is no longer available.

NOTE: Despite being invoked as “rmview”, this form of the command has no effect on any view, only on a specified VOB.

Caution

Incorrect results occur if a VOB loses synchronization with its views. Accordingly:

- Never remove a view with UNIX *rm*(1); always use the *rmview* command.
- If a view still exists, do not use `rmview -uuid` to delete records relating to it from any VOB. Make sure that the view need not be used again before using this command.

View UUIDs

Each view has a universal unique identifier, such as:

```
52000002.4ac711cb.a391.08:00:69:02:18:22
```

The listing produced by a `describe -long -vob` command includes the UUIDs of all views for which the VOB holds checkout records and derived object records.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

–for·ce Automatically responds "yes" to confirmation requests that *rmview* would otherwise make:

- Deleting a view storage directory: confirmation is needed to proceed if some elements are checked–out to the view. Proceeding has the effect of cancelling the checkouts: *rmview* removes the checkout records from the appropriate VOBs.
- Removing view-related records: confirmation is needed to proceed if the view still exists.

–tag *view-tag*
view-storage-dir-pname (mutually exclusive)

–tag specifies the view-tag of any view. In addition to removing the view storage area, *rmview* removes all relevant entries from the network's *view registry*.

view-storage-dir-pname specifies the top-level directory of the view storage area. Be sure that the current working directory is not anywhere within this view storage area.

–vob *pname-in-vob*

Specifies the VOB from which view-related records are to be removed. *pname-in-vob* can be the pathname of any object within the VOB. If you omit this option, *cleartool* uses the VOB containing the current working directory.

–uuid *view-uuid*

Specifies the view whose records are to be removed from a VOB.

EXAMPLES

- Delete the view storage area at */view_store/Rel2.vws*.

```
% cleartool rmview /view_store/Rel2.vws
```

- Delete the view storage area whose view-tag is *anneRel2*.

```
% cleartool rmview –tag anneRel2
```

- Delete the checkout and DO records for a deleted view from the current VOB. Suppress the confirmation prompt.

```
% cleartool rmview –force –uuid 249356fe.d50f11cb.a3fd.00:01:56:01:0a:4f
Removed references to VIEW "host2:/usr/vobstore/tut/old.vws"
        from VOB "/usr/hw".
```

SEE ALSO

cleartool subcommands: describe, lsview, mkview, mktag, rmtag, unregister
registry_ccase

NAME rmvob – remove a VOB storage directory

SYNOPSIS

rmvob [**–for·ce**] *vob-storage-dir-pname …*

DESCRIPTION

Deletes one or more VOB storage directories. Confirmation for each VOB is required, unless you use the –`force` option.

In addition to removing the VOB storage directory, *rmvob* removes all relevant entries from the network's *VOB registry*. However, *rmvob* does not unmount the VOB(s). Be sure to warn users and unmount (*cleartool umount*) a VOB before you delete its storage area!

If, before using *rmvob*, you do not unmount the VOB on all hosts where it is mounted, you must use the standard operating system commands *umount* and *rmdir* to reclaim the VOB mount point on each host.

CAUTION: Be sure that the current working directory is not within the VOB storage area that you are deleting.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, root user.
Locks: No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

–for·ce Suppresses the confirmation step.

vob-storage-dir-pname …
The pathnames of one or more VOB storage directories to be removed.

EXAMPLES

- Unmount and delete the VOB storage area */usr/vobstore/project.vbs* mounted on */vobs/project*.

```
% cleartool umount /vobs/project

% cleartool rmvob /usr/vobstore/project.vbs
Remove versioned object base "/usr/vobstore/project.vbs"?  [no] yes
Removed versioned object base "/usr/vobstore/project.vbs".
```

SEE ALSO

cleartool subcommands: mkvob, umount
filesys_ccase, registry_ccase

NAME rnpool – rename a VOB storage pool

SYNOPSIS

rnpool [**–vob** *pname-in-vob*] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
old-pool-name new-pool-name

DESCRIPTION

Renames a VOB storage pool. No data container in the pool is affected.

NOTE: This command does not work with pools created with the `mkpool -mount` command (which is no longer supported).

OPTIONS AND ARGUMENTS

Specifying the VOB. *Default:* Renames a storage pool in the VOB containing the current working directory.

–vob *pname-in-vob*
The VOB whose pool is to be renamed. *pname-in-vob* can be the pathname of any object within the VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Old and New Names. *Default:* None.

old-pool-name
new-pool-name
The name of an existing storage pool, and a new name for it. Compose the name(s) according to these rules:

- It must contain only letters, ideographs, digits, and the special characters underscore (_), period (.), and hyphen (-). The first character must not be a hyphen.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: pool creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, pool. See the "Permissions Checking" section of the *cleartool* manual page.

EXAMPLES

- Rename one of the current VOB's pools from *c_pool* to *c_source_pool*.

```
% cleartool rnpool –c "make pool name clearer" c_pool c_source_pool
Renamed pool from "c_pool" to "c_source_pool".
```

- List existing pools in the current VOB. Then, rename pool *do1* to *do_staged*.

```
% cleartool lspool –short
c_source_pool
cdft
ddft
do1
my_ctpool
sdft
% cleartool rnpool do1 do_staged
Renamed pool from "do1" to "do_staged".
```

SEE ALSO

cleartool subcommands: describe, chpool, lspool, mkpool, rmpool
profile_ccase, chmod(1)

NAME rntype – rename a type object

SYNOPSIS

rntype { **–elt·ype** | **–brt·ype** | **–lbt·ype** | **–att·ype** | **–hlt·ype** | **–trt·ype** | **–rpt·ype** }
[**–vob** *pname-in-vob*] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
old-type-name new-type-name

DESCRIPTION

Renames a type object in a VOB database. This effectively renames all instances of the type object, throughout the VOB. For example, renaming a branch type from *bugfix* to *rel1.3_fixes* effectively renames all existing *bugfix* branches to *rel1.3_fixes*.

RESTRICTION: A VOB cannot contain a branch type and a label type with the same name.

NOTE: Do not use this command to rename a particular branch of a particular element. For that purpose, use *chtype*.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type creator, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, type. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Type Object. *Default:* None.

–elt·ype (element type)
–brt·ype (branch type)
–lbt·ype (label type)
–att·ype (attribute type)
–hlt·ype (hyperlink type)
–trt·ype (trigger type)
–rpt·ype (replica type)
(mutually exclusive) Specifies the kind of type object to be renamed.

Specifying the VOB. *Default:* Renames a type in the VOB that contains the current working directory.

–vob *pname-in-vob*
The VOB whose type object is to be renamed. *pname-in-vob* can be any location within the VOB.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

Specifying the Old and New Names. *Default:* None.

old-type-name
new-type-name
The name of an existing type object, and a new name for it. For information on valid names, consult the manual page for the command that creates the type object. For example, consult the *mklbtype* manual page when renaming a label type.

EXAMPLES

- Rename a branch type from *rel2_bugfix* to *r2_maint*. First, show the version tree for *util.c* with the *lsvtree* command. Then rename the branch type, and show the version tree again.

```
% cleartool lsvtree –short util.c
util.c@@/main/1
util.c@@/main/rel2_bugfix
util.c@@/main/rel2_bugfix/1
util.c@@/main/3

% cleartool rntype –brtype rel2_bugfix r2_maint
Renamed type from "rel2_bugfix" to "r2_maint".

% cleartool lsvtree –short util.c
util.c@@/main/1
util.c@@/main/r2_maint
util.c@@/main/r2_maint/1
util.c@@/main/3
```

- Rename the element type of *msg.c* and *hello.c* from *text_file* to *source_file*. Use *grep*(1) to extract the element name/value from the output of the *describe* command. (Note warning about renaming a predefined type.)

```
% cleartool describe msg.c hello.c | grep 'element type'
  element type: text_file
  element type: text_file

% cleartool rntype –eltype text_file source_file
cleartool: Warning: Renaming a predefined object!
Renamed type from "text_file" to "source_file".

% cleartool describe msg.c hello.c | grep 'element type'
  element type: source_file
  element type: source_file
```

- Rename an attribute attached to a version of element *msg.c* from *TESTED* to *QAed*. Use *describe* to show the name/value association before and after the name change.

```
% cleartool describe –attr TESTED msg.c
msg.c@@/main/3
  Attributes:
    TESTED = "TRUE"

% cleartool rntype –attype TESTED QAed
Renamed type from "TESTED" to "QAed".

% cleartool describe –attr QAed msg.c
msg.c@@/main/3
  Attributes:
    QAed = "TRUE"
```

SEE ALSO

cleartool subcommands: describe, lstype, rmtype
profile_ccase

NAME

setcs – set the config spec of a view

SYNOPSIS

setcs [**–tag** *view-tag*] { **–cur·rent** | **–def·ault** | *file* }

DESCRIPTION

Changes the *config spec* of a view to the contents of a user-specified or system-default file, or causes the view's associated *view_server*(1M) process to flush its caches and reevaluate the current config spec. If the *working directory view* differs from the *set view* (established by the *setview* command), a warning message appears and the working directory view is reconfigured. See the *pwv* manual page for more on *view contexts*. See the *config_spec* manual page for a complete discussion of config specs.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the Kind of Change. *Default:* None.

–cur·rent Causes the *view_server* to flush its caches and reevaluate the current config spec, which is stored in file *config_spec* in the view storage directory. This includes:

- reevaluating *time rules* with non-absolute specifications (for example, `now`, `Tuesday`)
- reevaluating `–config` rules, possibly selecting different derived objects than previously
- rereading files named in `include` rules

–def·ault Resets the view's config spec to the contents of */usr/atria/default_config_spec*, the host's default config spec.

file Specifies an ASCII text file whose contents are to become the view's new config spec.

Specifying the View. *Default:* Reconfigures the current view.

–tag *view-tag*
The view-tag of any view; the view need not be active.

EXAMPLES

- Change the config spec of the current view to the contents of file *cspec_REL3*.
  ```
  % cleartool setcs cspec_REL3
  ```
- Change the config spec of the view whose view-tag is *jackson_vu* to the ClearCase default config spec.
  ```
  % cleartool setcs –tag jackson_vu –default
  ```
- Have the *view_server* of the current view reread its config spec.
  ```
  % cleartool setcs –current
  ```

SEE ALSO

cleartool subcommands: lsview, mkview, mktag, pwv
config_spec, view_server

NAME setview – create a process that is set to a view

SYNOPSIS

setview [**–log·in**] [**–exe·c** *cmd-invocation*] *view-tag*

DESCRIPTION

Creates a process that is *set* to the specified view. The new process is said to have a *set view* context. The process can execute a shell program (the default) or another program, as specified with a command option.

If you specify an inactive view — one whose *view-tag* does not appear in the local host's *viewroot* directory, */view* — a *startview* command is invoked implicitly to activate the view.

Once you set the view, you can take advantage of ClearCase's *transparency* feature: using standard pathnames to access version-controlled objects. The associated *view_server* process automatically resolves a standard pathname to an element into a reference to one of the element's versions. See the *pathnames_ccase* manual page for further details.

Using 'setview' in Interactive Mode

The shell command `cleartool setview` creates a subprocess. If you enter the *setview* command in interactive mode (at the `cleartool>` prompt), the new view is set in the current process. To push to a subprocess of an interactive *cleartool* process, use `setview -exec cleartool`.

View-Extended Naming and Set Views

Whether you have set a view or not, a view-extended pathname is interpreted with respect to the explicitly-named view. For example, */view/bugfix/usr/project/foo.c* always specifies the version of element *foo.c* selected by view *bugfix*.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Shell Startup Processing. *Default:* Does not read any shell startup file when starting a shell process.

–log·in If a shell process is started by *setview*, reads in your shell startup file:

SHELL Value	Startup Script
/bin/csh	*.login* in your home directory
/bin/sh	*.profile* in your home directory

No error occurs if the startup file is missing. Use this option to gain access to your personal aliases, environment variable settings, and so on.

Command to Execute in View Context. *Default:* A shell process is started, as indicated by your SHELL environment variable. If SHELL has a null value or is undefined, starts a Bourne shell (*/bin/sh*).

–exe·c *cmd-invocation*

Invokes the specified command line in view *view-tag*, instead of executing a shell. This command inherits the environment of your current process.

NAME setview – create a process that is set to a view

SYNOPSIS

setview [**–log·in**] [**–exe·c** *cmd-invocation*] *view-tag*

DESCRIPTION

Creates a process that is *set* to the specified view. The new process is said to have a *set view* context. The process can execute a shell program (the default) or another program, as specified with a command option.

If you specify an inactive view — one whose *view-tag* does not appear in the local host's *viewroot* directory, */view* — a *startview* command is invoked implicitly to activate the view.

Once you set the view, you can take advantage of ClearCase's *transparency* feature: using standard pathnames to access version-controlled objects. The associated *view_server* process automatically resolves a standard pathname to an element into a reference to one of the element's versions. See the *pathnames_ccase* manual page for further details.

Using 'setview' in Interactive Mode

The shell command `cleartool setview` creates a subprocess. If you enter the *setview* command in interactive mode (at the `cleartool>` prompt), the new view is set in the current process. To push to a subprocess of an interactive *cleartool* process, use `setview -exec cleartool`.

View-Extended Naming and Set Views

Whether you have set a view or not, a view-extended pathname is interpreted with respect to the explicitly-named view. For example, */view/bugfix/usr/project/foo.c* always specifies the version of element *foo.c* selected by view *bugfix*.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Shell Startup Processing. *Default:* Does not read any shell startup file when starting a shell process.

–log·in If a shell process is started by *setview*, reads in your shell startup file:

SHELL Value	Startup Script
/bin/csh	*.login* in your home directory
/bin/sh	*.profile* in your home directory

No error occurs if the startup file is missing. Use this option to gain access to your personal aliases, environment variable settings, and so on.

Command to Execute in View Context. *Default:* A shell process is started, as indicated by your SHELL environment variable. If SHELL has a null value or is undefined, starts a Bourne shell (*/bin/sh*).

–exe·c *cmd-invocation*

Invokes the specified command line in view *view-tag*, instead of executing a shell. This command inherits the environment of your current process.

Specifying the View. *Default:* None.

view-tag Any view-tag registered for the current network region. Use the *lsview* command to list registered view-tags.

EXAMPLES

- Create a shell process that is set to view *jackson_fix*, and run your shell startup script.

```
% cleartool setview –login jackson_fix
```

- Create a subprocess that is set to view *jackson_fix*, and run a script named */myproj/build_all.sh* in that process. Note that the command string must be enclosed in quotes.

```
cleartool> setview –exec "/myproj/build_all.sh" jackson_fix
```

- Set the current view to *jackson_old*, with the new process permanently in version-extended namespace. (Assumes `jackson_old@@` already exists.)

```
% cleartool setview jackson_old@@
```

SEE ALSO

cleartool subcommands: lsview, cd, pwv, mktag, shell, startview
pathnames_ccase, view_server

NAME shell – create a subprocess to run a shell or other program

SYNOPSIS

sh·ell | **!** [*command* [*arg* …]]

DESCRIPTION

Creates a subshell with the same view context as the current process. If the current process is *set* to one view, but the working directory view is different, *shell* uses the working directory view. (See the *pwv* manual page for more on this topic.)

The *shell* command is intended for use in *cleartool*'s interactive mode, at the `cleartool>` prompt. If you are using *cleartool* in single-command mode, there is no need for this command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Program to Run in Subprocess. *Default:* Runs the shell program indicated by your SHELL environment variable (or */bin/sh* if your environment does not include SHELL). The shell runs interactively until you exit from it.

command [*arg* …]
: Runs a non-interactive shell which, in turn, invokes the program *command*, (and, optionally, passes it one or more arguments). The subshell exits immediately after executing *command*.

EXAMPLES

- Create a subshell that is set to the same view as the *cleartool* process.

```
cleartool> shell
%
```

- Create a subshell, and run a command within it.

```
cleartool> ! head –2 /etc/passwd
sysadm:*:0:0:System V Administration:/usr/admin:/bin/sh
diag:*:0:996:Hardware Diagnostics:/usr/diags:/bin/csh
```

SEE ALSO

cleartool subcommands: pwv, setview
csh(1), sh(1)

NAME space – report on VOB disk space usage

SYNOPSIS

space [**–a·ll**] { **–avo·bs** | **–dir·ectory** *dir-pname …* | *pname-in-vob …*
| *vob-storage-dir-pname …* }

DESCRIPTION

Reports disk space usage for VOBs, or for non-ClearCase files or directories. The report for a VOB includes disk-usage information for the VOB database and for each storage pool. It also includes "extras", such as disk usage of the partition in which the VOB storage directory resides, and statistics on backup VOB databases left behind by invocations of *reformatvob*.

The report is organized by disk partition. The VOB's disk-usage statistics are reported both in absolute units (Mb) and as a percentage of the capacity of the disk partition containing the VOB storage directory.

Protection Errors

Certain data structures within a VOB storage directory are protected vigorously — for example, the *.identity* subdirectory. If you do not have permission to examine a file or subdirectory, *space* displays a `Permission denied` message and does not include that item in its calculations.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Report Format. *Default:* In a report on a VOB storage directory, files whose names begin with a dot (`.`) character are not listed; neither are files known to be small. (The contribution of these files is still included in the disk-usage total.)

–a·ll Lists dot-files (*.identity*, *.pid*, and so on) and small files individually. See "Protection Errors" above.

Specifying the Data Structures. *Default:* None.

–avo·bs Reports on all VOBs active (mounted) on the local host. (If environment variable CLEARCASE_AVOBS is set to a colon-separated list of VOB-tags, this set of VOBs is used instead.)

pname-in-vob …
One or more pathnames, each of which specifies a VOB to be included in the report. Any pathname within the VOB is valid.

vob-storage-dir-pname …
One or more pathnames of VOB storage directories.

–dir·ectory *dir-pname …*
One or more pathnames, specifying locations that are *not* under a VOB-tag (VOB mount point).

EXAMPLES

- Report on all VOBs currently mounted on the local host.

```
% cleartool space –avobs
 Use(Mb)  %Use  Directory
     1.5    0%  VOB database /net/helium/usr1/vobstorage/demo_src_vob/db
     0.0    0%  cleartext pool /net/helium/usr1/vobstorage/demo_src_vob/c/cdft
     0.0    0%  derived object pool /net/helium/usr1/vobstorage/demo_src_vob/d/ddft
    31.0    3%  source pool /net/helium/usr1/vobstorage/demo_src_vob/s/sdft
    44.2    4%  VOB database /net/helium/usr1/vobstorage/raima_vob/db
     2.9    0%  cleartext pool /net/helium/usr1/vobstorage/raima_vob/c/cdft
    88.7    8%  derived object pool /net/helium/usr1/vobstorage/raima_vob/d/ddft
     5.5    0%  source pool /net/helium/usr1/vobstorage/raima_vob/s/sdft
      :
      :
    13.6    1%  source pool /net/helium/usr1/vobstorage/public_vob/s/sdft
    12.1    1%  VOB database /net/helium/usr1/vobstorage/motif_vob/db
     5.6    0%  cleartext pool /net/helium/usr1/vobstorage/motif_vob/c/cdft
     0.0    0%  derived object pool /net/helium/usr1/vobstorage/motif_vob/d/ddft
    43.4    4%  source pool /net/helium/usr1/vobstorage/motif_vob/s/sdft
--------  ----  -------------------------------------------------------
   555.3   50%  Subtotal
   919.8   82%  Filesystem /tmp_mnt/net/helium/usr1 (capacity 1115.1 Mb)

 Use(Mb)  %Use  Directory
     0.0    0%  unknown item /net/viewpnt/usr1/vobstorage/atria_vob/event_scrubber_params
   214.3   22%  old VOB database /net/viewpnt/usr1/vobstorage/atria_vob/db.01.04
   363.0   38%  VOB database /net/viewpnt/usr1/vobstorage/atria_vob/db
   169.1   18%  source pool /net/viewpnt/usr1/vobstorage/atria_vob/s/sdft
     0.9    0%  source pool /net/viewpnt/usr1/vobstorage/atria_vob/s/staged_includes
--------  ----  -------------------------------------------------------
   747.3   78%  Subtotal
   883.8   92%  Filesystem /tmp_mnt/net/viewpnt/usr1 (capacity 961.2 Mb)
      :
      :
Total usage for vob "/vobs/demo_src" 32.5 Mb
Total usage for vob "/vobs/atria" 1373.3 Mb
      :
      :
Total usage for vob "/vobs/raima" 141.3 Mb
Total usage for vob "/vobs/design" 4.3 Mb
Total usage for vob "/vobs/doc" 277.3 Mb
Total usage for vob "/vobs/int" 3.6 Mb
Total usage for vob "/vobs/public" 31.1 Mb
Total usage for vob "/vobs2/bob.c++" 0.9 Mb
Total usage for vob "/usr/var/tmp/btest" 2.2 Mb
Total usage for vob "/vobs/scd" 4.3 Mb
```

- Report space usage for a given VOB showing all files (long report).

```
% cleartool space –all msg.c
 Use(Mb)  %Use  Directory
     0.0    0%  unknown item /net/reach/usr/var/tmp/scd/tut.vbs/.identity
     0.0    0%  VOB identifier /net/reach/usr/var/tmp/scd/tut.vbs/vob_oid
     0.0    0%  replica identifier /net/reach/usr/var/tmp/scd/tut.vbs/replica_uuid
     0.0    0%  unknown item /net/reach/usr/var/tmp/scd/tut.vbs/.pid
     1.4    0%  VOB database /net/reach/usr/var/tmp/scd/tut.vbs/db
     0.0    0%  cleartext pool /net/reach/usr/var/tmp/scd/tut.vbs/c/cdft
     0.0    0%  derived object pool /net/reach/usr/var/tmp/scd/tut.vbs/d/ddft
```

```
     0.0    0%  source pool /net/reach/usr/var/tmp/scd/tut.vbs/s/sdft
--------  ----  ---------------------------------------------------------
     1.4    0%  Subtotal
   161.4   63%  Filesystem /usr (capacity 256.3 Mb)

Total usage for vob "msg.c" 1.4 Mb
```

- Report space usage for a given VOB using the default output.

```
% cleartool space msg.c
 Use(Mb)  %Use  Directory
     1.4    0%  VOB database /net/reach/usr/var/tmp/scd/tut.vbs/db
     0.0    0%  cleartext pool /net/reach/usr/var/tmp/scd/tut.vbs/c/cdft
     0.0    0%  derived object pool /net/reach/usr/var/tmp/scd/tut.vbs/d/ddft
     0.0    0%  source pool /net/reach/usr/var/tmp/scd/tut.vbs/s/sdft
--------  ----  ---------------------------------------------------------
     1.4    0%  Subtotal
   161.4   63%  Filesystem /usr (capacity 256.3 Mb)

Total usage for vob "msg.c" 1.4 Mb
```

SEE ALSO

cleartool subcommands: mkvob, reformatvob
env_ccase, df(1M), du(1M)

NAME startview – start or connect to a view_server process

SYNOPSIS

startview *view-tag …*

DESCRIPTION

Prerequisite: The view being started must already have a 'view-tag' in the network's 'view_tag' registry file. See the 'mkview' and 'mktag' manual pages.

Enables processes on the local host to access a view, by:

- Establishing an RPC connection between the local host's *MVFS* (ClearCase *multiversion file system*) and the view's *view_server* process.
- Creating a *view-tag* entry in the local host's *viewroot* directory. If a *view_server* process is not already running, *startview* invokes one on the host where the view storage area physically resides.

The default name of the viewroot directory is */view*. (See the *init_ccase* manual page for more information.) Thus, starting a view that has been registered under view-tag *main* would create directory entry */view/main*. After this directory entry is created, any process on the local host can access the view through view-extended pathnames.

The view's view-tag must already be registered, which is accomplished either at view creation time (with a *mkview* command) or subsequently (with `mktag -view`).

When to Use 'startview'

Both *mkview* and *mktag* implicitly perform a *startview*. Furthermore, the *setview* command also performs a *startview*, if necessary. Therefore, it is rarely necessary to invoke *startview* explicitly. Typically, *startview* is used to establish view-extended naming access, without creating a process that is set to the view (as happens with *setview*). There are two main cases:

- Because *mkview* and *mktag* perform a *startview* on the local host only, remote users who want only view-extended naming access to the view must use *startview*.
- After your system has been stopped and restarted (see "Examples" below), both local and remote users can use *startview* to re-establish view-extended naming access to a view.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the View. *Default:* None.

view-tag … One or more view-tags currently listed in the local host's *view-tags file*.

EXAMPLES

- View *anne_Rel2* is registered, but its *view_server* process went down in a system crash. Restart view *anne_Rel2*, and make it the working directory view.

```
% cleartool startview anne_Rel2
% cd /view/anne_Rel2/usr/hw
```

- Create a view on the local host, and establish view-extended naming access to the view on *host3*.

```
% cleartool mkview –tag mainRel2 /view_store/mainRel2.vws
Created view.
Host-local path: host2:/view-store/mainRel2.vws
Global path:     /net/host2/view-store/mainRel2.vws
It has the following rights:
User : anne     : rwx
Group: dev      : rwx
Other:          : r-x
% rsh host3 cleartool startview mainRel2
```

SEE ALSO

cleartool subcommands: lsview, setview, mkview, mktag
filesys_ccase, init_ccase, registry_ccase, view_server

NAME umount – deactivate a VOB

SYNOPSIS

umount { *vob-tag* | **–all** }

DESCRIPTION

Deactivates one or more VOBs on your host by unmounting them as UNIX-level file systems. A VOB is activated on a host by mounting it as a file system of type MVFS (ClearCase's *multiversion file system* type). The *VOB-tag* by which an individual VOB is referenced is the same as the full pathname to its mount point.

Note that *umount* has no impact on a VOB's entries in the *vob_object* and *vob_tag* registry files.

Unmounting of Public and Private VOBs

The *root* user can unmount any VOB, public or private; other users can unmount only the private VOBs that they, themselves own.

See the *mkvob* manual page for a discussion of *public* and *private* VOBs.

Unmounting All VOBs

The *root* user can use `cleartool umount -all` to unmount all *public* VOBs listed in the VOB registry.

PERMISSIONS AND LOCKS

Permissions Checking: See "Unmounting of Public and Private VOBs" above. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

Specifying the VOB(s). *Default:* None.

vob-tag Unmounts the VOB with this *VOB-tag*, which you must specify exactly as it appears in the *vob_tag* registry file.

–all (*root* user only) Unmounts all public VOBs listed in the VOB registry.

IMPLEMENTATION NOTE

umount calls the standard *umount*(1M) command.

EXAMPLES

- Unmount the VOB storage directory that is registered with VOB-tag */vobs/Rel4*.

```
% cleartool umount /vobs/Rel4
```

- Unmount all VOBs registered with public VOB-tags.

```
% su                                    (become 'root' user)
# cleartool umount -all                 (unmount all public VOBs)
```

SEE ALSO

cleartool subcommands: umount, lsview, lsvob, register, mktag, mkview, mkvob, registry_ccase, umount(1M)

NAME uncheckout – cancel a checkout of an element

SYNOPSIS

uncheck·out | unco [–kee·p | –rm] *pname …*

DESCRIPTION

Cancels a checkout for one or more elements, deleting the checked-out version. Any meta-data (for example, attributes) that you attached to a checked-out version is lost. The view reverts to selecting a checked-in version of each element.

The `checkout version` event record for each element is removed from its VOB's database. (There is no such such thing as an "uncheckout" event record.)

If you checked out a file under an alternate name (`checkout -out`), you cannot use the alternate name to *uncheckout* the file — you must use the element name listed by `ls -vob_only`.

Cancelling a Checkout in an Inaccessible View

You can cancel another view's checkout by using a view-extended pathname to the element. But if the other view is no longer accessible (for example, it was deleted accidentally), a view-extended pathname will not work. Instead:

1. Use `describe -long -vob` to determine the view's unique identifier (*uuid*).
2. For each VOB that may have been accessed with the view, use the `rmview -uuid` command to remove all of that view's checkout records from the VOB. (There is no way to selectively cancel checkouts for an inaccessible view.)

Cancelling a Directory Checkout

If you cancel a directory's checkout after changing its contents, the changes made with *rmname*, *mv*, and *ln* are lost. Any new elements that were created (with *mkelem* or *mkdir*) become *orphaned*; such elements are moved to the VOB's *lost+found* directory, stored under names of this form:

element-name.UUID

uncheckout displays a message in such cases:

```
cleartool: Warning: Object "foo.c" no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory as
 "foo.c.5f6815a0a2ce11cca54708006906af65".
```

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: version creator, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Handling of the View-Private File. *Default: uncheckout* prompts you to decide whether to preserve the checked-out version in a view-private file:

```
Save private copy of "util.c"?  [yes]
```

A `yes` answer is equivalent to specifying the `–keep` option; a `no` answer is equivalent to specifying the `–rm` option.

–kee·p Preserves the contents of the checked-out version under a file name of the form *element-name**.keep*** (or, to prevent name collisions, *element-name**.keep.1***, *element-name**.keep.2***, and so on).

–rm Does not preserve the contents of the checked-out version. Thus, any edits that had been made to the checked-out version are lost.

Specifying the Element. *Default:* None.

pname … One or more pathnames, each of which specifies an element. The checkout in the current view is cancelled, unless you use a view-extended pathname to specify another view.

NOTE: Avoid using a version–extended pathname — for example, you cannot use `hello.c@@/main/sub1` to cancel another view's checkout on the *sub1* branch of element *hello.c*.

EXAMPLES

- Cancel the checkout of file element *util.c*.

```
% cleartool uncheckout util.c
Save private copy of "util.c"?  [yes] no
Checkout cancelled for "util.c".
```

- Cancel the checkout of file *hello.h* in the *jackson_fix* view, and delete the view-private copy.

```
% cleartool uncheckout –rm /view/jackson_fix/usr/hw/src/hello.h
Checkout cancelled for "/view/jackson_fix/usr/hw/src/hello.h".
```

- Cancel the checkout of directory *subd* after creating a new element named *conv.c*. Note that the element is moved to the VOB's *lost+found* directory.

```
% cleartool uncheckout subd
Object moved to vob lost+found directory as
        "conv.c.3d90000112fc11cba70e0800690605d8".
Checkout cancelled for "subd".
```

SEE ALSO

cleartool subcommands: checkin, checkout, mkview, unreserve

NAME unlock – unlock an object

SYNOPSIS

- Unlock entire VOB:

  unlock [**–c** *comment* | **–cq** | **–cqe** | **–nc**] **–vob** { *pname-in-vob* | *vob-storage-dir-pname* }

- Unlock VOB storage pool:

  unlock [**–c** *comment* | **–cq** | **–cqe** | **–nc**] [**–vob** *pname-in-vob*] **–poo·l** *pool-name …*

- Unlock element or branch:

  lock [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname …*

- Unlock type object:

  lock [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
  { **–elt·ype** | **–brt·ype** | **–att·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–rpt·ype** }
  [**–vob** *pname-in-vob*] *type-name …*

DESCRIPTION

Removes an existing *lock* from an entire VOB, or from one or more file system objects, *type objects*, or *VOB storage pools*. See the *lock* manual page for a description of ClearCase locks.

PERMISSIONS AND LOCKS

Kind of Object	Users Permitted to Unlock the Object
type object	type creator, VOB owner, root user
storage pool	VOB owner, root user
VOB	VOB owner, root user
element	element owner, VOB owner, root user
branch	branch creator, element owner, VOB owner, root user

Even if you have permission to execute this command, it fails if an entire-VOB lock has been placed on the VOB containing the object.

OPTIONS AND ARGUMENTS

See the *lock* manual page for a description of the options to the *unlock* command.

EXAMPLES

- Unlock the label types *REL1* and *REL2*.

```
% cleartool unlock –lbtype REL1 REL2
Unlocked label type "REL1".
Unlocked label type "REL2".
```

SEE ALSO

cleartool subcommands: lock, lshistory, lslock, lspool, lstype, protect
profile_ccase

NAME unregister – remove an entry from the vob_object or view_object registry file

SYNOPSIS

- Unregister a VOB:

  unreg·ister −vob { **−uui·d** *uuid* | *vob-storage-dir-pname* }

- Unregister a view:

  unreg·ister −view { **−uui·d** *uuid* | *view-storage-dir-pname* }

DESCRIPTION

Removes the entry for a particular VOB or view from the network's *vob_object* registry file or from the *view_object* registry file. The *unregister* command does not affect VOB-tag or view-tag registry entries, and it does not affect the contents of the physical storage directories. See the *registry_ccase* manual page for a discussion of registry files.

Note that removing a VOB or view storage directory with an operating system command (`rm -rf`, for example), rather than with *rmvob* or *rmview*, fails to unregister the VOB/view. In this case, you must use the `−uuid` option to unregister the associated storage directory (and use *rmtag* to remove relevant tag entries, if any still exist).

Other Commands that Affect Storage Registries

The *mkview* and *mkvob* commands automatically add an entry to the appropriate registry; the *rmview* and *rmvob* commands remove registry entries (and the actual storage directories as well). You can use the *register* command to update an existing entry, or to re-register a VOB or view that has been unregistered.

The *reformatvob* command updates a VOB's *object registry* entry (or creates one, if necessary), but does not affect its *tag registry* entries.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

View/VOB Specification. *Default:* None. — you must indicate whether you are unregistering a view or a VOB, and must specify either its storage directory or its unique identifier (*UUID*).

−vob *vob-storage-dir-pname*
−vob −uui·d *vob-uuid*
: Use either of the forms to specify the VOB whose *vob_object* registry entry is to be deleted.

−view *view-storage-dir-pname*
−view −uui·d *view-uuid*
: Use either of the forms to specify the view whose *view_object* registry entry is to be deleted.

EXAMPLES

- Unregister a VOB storage directory.

```
% cleartool unregister –vob /vobstore/vob2.vbs
```

- Unregister a view storage directory.

```
% cleartool unregister –view /view_store/view5.vws
```

- Using the `–uuid` option, unregister a VOB storage directory that was deleted with `rm -rf` instead of *rmvob*. In this example, the *VOB replica UUID* (do not use the *VOB family UUID*) is found in the output from `lsvob -long`. After unregistering the storage directory, remove the VOB-tag. If the VOB has tag registry entries for more than one network region, the `–all` option removes them all.

```
% cleartool lsvob -long /vobs/src                    (find the VOB replica uuid)
Tag: /vobs/src
  Global path: /net/neptune/vobstore/src.vbs
  :
Vob replica uuid: cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3

% ls /net/neptune/vobstore/src.vbs                   (verify storage directory was removed)
UX:ls: ERROR: Cannot access /net/neptune/vobstore/src.vbs: No such file or directory

% cleartool unregister –vob –uuid cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3
% cleartool rmtag –vob –all /vobs/src
```

- As in the previous example, unregister a removed but still registered VOB storage directory. In this example, the VOB-tag has already been removed. Therefore, we use the */usr/adm/atria/log/scrubber_log*, not *lsvob*, to find the VOB replica UUID. (*lsvob* lists only VOBs that have registered VOB-tags.) The *scrubber* utility, which runs nightly by default, reports the required UUID in an error message after failing to find the registered storage directory.

```
% cat /usr/adm/atria/log/scrubber_log
  :
05/27/94 04:30:58 scrubber: Error: Unable to get VOB tag registry information for
  replica uuid "cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3": ClearCase object not found
05/27/94 04:30:58 scrubber: Error: unable to access VOB neptune:/vobstore/src.vbs:
  ClearCase object not found
05/27/94 04:30:58 scrubber: Warning: skipping VOB neptune:/vobstore/src.vbs due to earlier errors
  :
% cleartool unregister –vob –uuid cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3
```

FILES

/usr/adm/atria/rgy/vob_object
/usr/adm/atria/rgy/view_object

SEE ALSO

cleartool subcommands: register, mkvob, mkview, mktag, register, mount, umount
registry_ccase

NAME unreserve – change a reserved checkout to unreserved

SYNOPSIS

unres·erve [**–vie·w** *view-storage-dir-pname*] [**–c** *comment* | **–cq** | **–cqe** | **–nc**] *pname* …

DESCRIPTION

Changes the "checkout status" of a checked–out version of an element to *unreserved*. An `unreserve checkout of version` event record is written to the VOB database.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, root user. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch. See the "Permissions Checking" section of the *cleartool* manual page.

OPTIONS AND ARGUMENTS

Specifying the View. *Default:* The current view's checkout is changed (unless you specify an element with a view-extended pathname).

–vie·w *view-storage-dir-pname*
Specifies the view whose checkout is to be changed. The view must be specified in *hostname*:*pathname* form, as listed by `lscheckout -l`. For example:

```
-view jupiter:/usr/people/jones/test/jones_view
```

Specifying the Elements. *Default:* None.

pname … One or more pathnames, each of which specifies an element. The checkout in the current view is changed, unless you use a view-extended pathname to specify another view.

Event Records and Comments. *Default:* Creates one or more event records, with commenting controlled by your home directory's *.clearcase_profile* file (default: –nc). See "Comment Handling" in the *cleartool* manual page. Comments can be edited with *chevent*.

–c *comment* , **–cq** , **–cqe** , **–nc**
Overrides the default with one of ClearCase's standard comment options.

EXAMPLES

- Change the checkout status of an element to unreserved.

```
% cleartool unreserve util.c
Changed checkout to unreserved for "util.c" branch "/main".
```

- Change the checkout status of an element in another view to unreserved. Note that the view's storage area is on a remote host.

```
% cleartool unreserve –view oxygen:/usr/home/jackson/ccviews/fix.vws hello.c
Changed checkout to unreserved for "hello.c" branch "/main".
```

- Checkout an element, check its status, then change its status to unreserved.

```
% cleartool co –nc edge.sh
Checked out "edge.sh" from version "/main/1".
```

```
% cleartool lsch edge.sh
08-Dec.12:17  jackson  checkout version "edge.sh" from /main/1 (reserved)
% cleartool unreserve edge.sh
Changed checkout to unreserved for "edge.sh" branch "/main".
```

SEE ALSO

cleartool subcommands: checkin, checkout, lscheckout, reserve, uncheckout
profile_ccase

NAME winkin – wink-in one or more derived objects to a view

SYNOPSIS

winkin [**–out** *pname*] *DO-pname* …

DESCRIPTION

This command enables you to access the data of any existing DO, even if it does not match your view's build configuration (and, thus, would not be winked-in by a *clearmake* build). Note that you cannot access a DO's file system data directly, using a VOB-extended pathname, such as *hello@@21-Dec.16:18.397*; instead, you must wink it in to a view, and then access it using that view.

Effect on View-Resident DO Data Containers

If you specify a shared DO while working in the view where it was originally built, and there is still a view-resident data container for the DO in that view, then the view-resident data container is *scrubbed*, and your view will access the shared data container in VOB storage. This is equivalent to executing a *view_scrubber* command.

If you specify an unshared DO in your view, it is promoted to the VOB and then winked-in. The view-resident data container is scrubbed, and your view will access the shared data container in VOB storage. This is equivalent to executing a *view_scrubber -p* command.

Note that *view_scrubber* is more efficient when a large number of DOs are to be processed in this way.

PERMISSIONS AND LOCKS

No special permissions are required at the VOB database level; no locks apply. At the file system level, you must have "read" permission on the DO to be winked-in. If you are overwriting an existing DO in your view (perhaps one that was winked-in previously), you must have "write" permission on the existing DO. See the CLEARCASE_BLD_UMASK discussion in the *clearmake* manual page.

OPTIONS AND ARGUMENTS

Specifying an Alternative Pathname. *Default:* A derived object is winked-in to your view at the pathname you specify with a *DO-pname* argument, minus any DO-ID. For example, if you specify the DO-pname *../src/hello@@21-Dec.16:18.397*, then by default, it is winked-in at pathname *../src/hello*. Any object at the destination pathname is overwritten, subject to standard permissions-checking. ("Overwriting" a shared DO just decrements its reference count — no file system data is actually deleted.)

–out *pname* An alternative pathname at which to wink-in the DO. You must specify exactly one DO in this case.

- If *pname* is a directory, then the DO is winked-in to that directory, with the same leaf name as the original DO.
- Otherwise, *pname* is treated as a file name.

In either case, an error occurs if an object already exists at the destination.

NOTE: You must use `–out` if you specify another view's DO using a view-extended pathname, and you intend to wink-in the DO to your own view.

Specifying the Derived Object. *Default:* None.

DO-pname ...

One or more pathnames that specify derived objects. A standard pathname names a DO in the current view; you can also use a view-extended pathname and/or a VOB-extended pathname:

```
/view/george/usr/hw/hello                    (view–extended pathname)
hello@@21-Dec.16:18.397                      (VOB–extended pathname, including DO–ID)
/view/george/usr/hw/hello@@05-Jan.09:16:788  (combination)
```

EXAMPLES

- Wink-in to your view a DO that appears in another view. The `–out` option is required in this case.

```
% cleartool winkin –out . /view/george/usr/hw/hello.o
Winked in derived object "hello.o"
```

- Wink-in a DO, using its DO-ID, and saving it under another file name.

```
% cleartool lsdo hello
02-Mar.20:02   "hello@@02-Mar.20:02.376"
01-Mar.09:06   "hello@@01-Mar.09:06.365"
% cleartool winkin –out hello.March1 hello@@01–Mar.09:06.365
Promoting unshared derived object "hello@@01-Mar.09:06.365"
Winked in derived object "hello.March1"
```

- Create a new derived object, then migrate it to VOB storage.

```
% clearmake
        cc -c hello.c
        cc -c util.c
        cc -o hello hello.o util.o
% cleartool winkin hello
Promoting unshared derived object "hello"
Winked in derived object "hello"
```

SEE ALSO

clearmake, derived_object, scrubber, view_scrubber

NAME xdiff – compare versions of a text-file element or a directory graphically

SYNOPSIS

xdiff [**–tin·y**] [**–hst·ack** | **–vst·ack**] [**–opt·ions** *pass-through-opts*]
[**–pre·decessor**] *pname* …

DESCRIPTION

xdiff creates a child process in which *xcleardiff*, the compare/merge utility in the ClearCase GUI, uses a separate window to display the differences among the specified files or directories. *xdiff* waits for the child process to exit (unlike *diff*).

xdiff uses *type managers* in the same way as the *diff*, except that it invokes the *xdiff* method (linked to *xcleardiff* for all predefined type managers).

Scrolling

You can scroll the individual subwindows (*difference panes*) of the *xcleardiff* window either independently or synchronously. All difference panes have a "locked/unlock" toggle button, which changes back and forth between `L` and `U` when you click it. When you use a vertical or horizontal scrollbar, all locked panes (with `L` showing) scroll together. When you click the "Previous Diff", "Next Diff", or "Current Diff" button, the difference panes resynchronize.

OPTIONS AND ARGUMENTS

The syntax of the *xdiff* command is the same as that for *diff*, with these exceptions:

- The `–window` option is not supported; *xdiff* always creates a new window.
- The mutually exclusive listing format options `–serial_format`, `–diff_format`, and `–columns` are not supported.
- The `–hstack` and `–vstack` options are added to support horizontal (side by side) and vertical stacking of the difference panes. The default is horizontal.

EXAMPLES

- Display the differences between versions of a source file in two different views.

```
% cleartool xdiff util.c /view/jackson_old/usr/hw/src/util.c
```

- Compare a specified version of *cm_add.c* to its predecessor.

```
% cleartool xdiff –predecessor cm_add.c@@/main/2
```

SEE ALSO

cleartool subcommands: diff, merge, xmerge
cleardiff, xcleardiff

NAME xlsvtree – list version tree of an element graphically

SYNOPSIS

xlsvt·ree [**–a·ll**] [**–nco**] [**–nme·rge**] [**–opt·ions** *pass-through-options*] *pname …*

DESCRIPTION

Invokes a *vtree browser* to display an element's version tree graphically. *cleartool xlsvtree* provides an interface to the *xlsvtree* utility program, which is a restricted version of *xclearcase*, the main ClearCase graphical interface program. (The commands *xlsvtree* and *cleartool xlsvtree* are functionally identical. The two invocation alternatives are provided as a convenience.)

A separate vtree browser is invoked for each element you specify as an argument.

OPTIONS AND ARGUMENTS

Controlling Which Versions Are Displayed. *Default:* The vtree browser displays all "significant" versions: versions with labels, versions that are branchpoints, and versions that are hyperlink endpoints.

–a·ll Displays all versions of the element. In addition, `–all` annotates each labeled version with its complete list of version labels. (By default, only the first five labels are displayed for any single version, followed by `...` when there are more than five.)

–nco Excludes checked-out versions from the display. The predecessor of a checked-out version is also excluded, unless there is another reason to include it (for example, it has a version label).

–nme·rge Excludes versions that have merge arrows.

xclearcase Options. *Default:* None.

–options *pass-through-options*

Specifies one or more *xclearcase* command options that are not directly supported on the *cleartool xlsvtree* command line. In particular, *xclearcase* accepts all the standard X Toolkit command-line options (for example, `–display` and `–geometry`), as described in the *X*(1) manual page. Quote the option string if it includes white space.

NOTE: When invoking *xclearcase* directly with *xlsvtree* (as opposed to *cleartool xlsvtree*), omit the `–options` keyword.

Specifying Elements to Browse. *Default:* None. You must specify at least one element.

pname … One or more elements. A vtree browser comes up for each element.

EXAMPLES

- Invoke an *xclearcase* vtree browser to display a file element's version tree. Show all versions.

```
% cleartool xlsvtree –all util.h
```

SEE ALSO

cleartool subcommands: lsvtree, lshistory
xclearcase, schemes

NAME xmerge – merge versions of a text-file element or a directory graphically

SYNOPSIS

xmerge { **–to** *contrib-&-result-pname* | **–out** *output-pname* } [**–bas·e** *pname*]
[**–nda·ta** | **–nar·rows**] [**–rep·lace**] [**–abo·rt** | **–qal·l**]
[**–opt·ions** *pass-through-options*] [**–c** *comment* | **–cq** | **–cqe** | **–nc**]
[**–tin·y**] [**–hst·ack** | **–vst·ack**] [**–ins·ert** | **–del·ete**]
{ **–ver·sion** *contrib-version-selector …* | *contrib-pname …* }

DESCRIPTION

xmerge creates a child process in which *xcleardiff*, ClearCase's graphical compare/merge utility, uses a separate window to perform a merge. *xmerge* waits for the child process to exit.

xmerge uses *type managers* in the same way as the *merge* command, except that it invokes the *xmerge* method (linked to *xcleardiff* for all predefined type managers).

OPTIONS AND ARGUMENTS

The syntax of the *xmerge* command is the same as that for *merge*, with these exceptions:

- The `–window` option is not supported; *xmerge* always creates a new window.
- The mutually exclusive listing format options `–serial_format`, `–diff_format`, and `–columns` are not supported.
- The `–hstack` and `–vstack` options are added to support horizontal (side by side) and vertical stacking of the merge contributor files. The default is horizontal, with the base contributor on the left.

EXAMPLES

- Merge the version of file *msg.c* in the current view with the most recent version on the *bugfix* branch. Determine the base file automatically, and record the merge with merge arrows.

```
% cleartool xmerge –to msg.c msg.c@@/main/bugfix/LATEST
```

- Merge into the version of file *util.c* in the view the most recent versions on the *rel2_bugfix* and *motif* branches. Suppress the creation of merge arrows.

```
% cleartool xmerge –to util.c –narrows \
    –version /main/rel2_bugfix/LATEST /main/motif/LATEST
```

SEE ALSO

cleartool subcommands: merge, findmerge, diff, xdiff
xcleardiff, xclearcase

NAME non_cleartool_divider – separator page

DESCRIPTION

FOR POSITION ONLY

THIS PAGE TO BE REPLACED

BY A FULL-PAGE RUBYLITH AND

"Non-cleartool Manual Pages"

FOR POSITION ONLY

BLANK PAGE WITH A

FULL-PAGE RUBYLITH

NAME abe – audited build executor / server for ClearCase distributed build

DESCRIPTION

This program is started by 'clearmake' when needed; it should never be run manually.

abe, the *audited build executor*, is a server process automatically invoked by *clearmake* to control and audit execution of a build script during a distributed build.

The first time it dispatches a build script to a host, *clearmake* starts an *abe* process there, using a standard remote-shell command. Subsequent build scripts dispatched to the same host may get executed by the same *abe* process, or by a different one.

Build Hosts File

Hosts for a distributed build are selected from the *build hosts file* of the user who executes the *clearmake* command. (See the *bldhost* manual page.) A host can be listed several times in the build hosts file, in which case several independent *abe* processes are invoked.

BUILD SCRIPT PROCESSING

An *abe* process starts by setting the same view as the calling *clearmake*. It executes a build script dispatched to it in much the same way as *clearmake* — each command in a separate shell process.

The build script has already had all its make macros expanded by the calling *clearmake*; but environment variables are expanded by the shell process in which a build command runs. This environment combines the *abe*'s startup environment and the entire environment of the calling *clearmake*. Where there are conflicts (for example, SHELL and PATH), the *abe* setting prevails. To this environment is added:

- Special make macros, such as MAKEFLAGS, MAKEARGS, and (in *smake*(1) compatibility mode only) MFLAGS. These are needed in case the build script invokes *clearmake* recursively.
- Macros assigned in a build options spec (see the *clearmake.options* manual page) or on the *clearmake* command line. These settings are always placed in the build script's environment; they override, if necessary, settings in the environment of the calling *clearmake* and/or settings in the *abe* startup environment.

The *stdout* and *stderr* output produced by build scripts is sent back to *clearmake*, which stores it in a temporary file. When the build script terminates, *clearmake* outputs its accumulated terminal output.

abe returns the exit status of the build script to the calling *clearmake*, thus indicating if the build succeeded or failed. In the "success" case, *abe* creates derived objects and configuration records.

Failure Modes

Certain conditions can interfere with an *abe* process, causing a target rebuild to fail:

- remote login is disabled on a particular host, preventing an *abe* process from being started
- *clearmake*'s view could not be accessed on the remote host

SEE ALSO

clearmake, config_spec, exec(1), rsh(1), setuid(2)
clearmake.options, bldhost, bldserver.control

NAME albd_server – location broker daemon / ClearCase master server

SYNOPSIS

invoked by ClearCase startup script at system startup time

DESCRIPTION

Each ClearCase host runs an *albd_server* process (Atria location broker daemon), which plays a pivotal role in ClearCase's client-server architecture. *albd_server* is a "master server", which starts up and dispatches messages to other servers:

db_server	VOB database server, short-lived
vob_server	VOB data storage server, long-lived
vobrpc_server	remote-access VOB database and data storage server, long-lived
promote_server	derived object data storage server, short-lived
view_server	view server, long-lived

A ClearCase client program sends a request to an *albd_server* process (often, running on another host) to find the port (socket address) of one of the servers listed above. Thereafter, the client communicates directly with the specific server. If necessary, *albd_server* starts up the server before passing its port number back to the client.

STARTING THE LOCATION BROKER

albd_server is started by the ClearCase startup script at system startup time. (See the *init_ccase* manual page for details.) Never invoke *albd_server* directly — only through the startup script.

SERVICES DATABASE

The ClearCase installation procedure creates an *albd_server* entry in a host's local *services*(4) database, */etc/services*:

```
# Location Broker Server
#
albd    371/udp
```

(If an NIS *services* map exists, the installation procedure advises the installer to update this map, if necessary.)

When it begins execution, *albd_server* looks itself up in its host's services database (file or NIS map). If the lookup returns the standard port number, 371, it creates an empty flag file, */usr/adm/atria/.albd_well_known_port*. ClearCase client programs on a host use this flag file to avoid lookups in the services database: if the file exists, a client uses the standard port number to contact *albd_server* processes throughout the network; otherwise, it must look up the `albd` service in the services database. Note that this scheme requires that the `albd` service be registered at the same port number throughout the network.

ALBD_SERVER CONFIGURATION FILE

Do not modify this configuration file, except under explicit instructions from Customer Support.

albd_server reads configuration file */usr/atria/config/services/albd.conf* during startup, to determine which services to provide. Lines that begin with `#` are comments, as are empty lines. All other lines must contain the white-space-separated fields described below. The character `-` in a field indicates "not applicable" or "use default".

Number — RPC program number for the service.

Version — RPC version number for the service.

Protocols — Comma–separated, no–white–space list of protocols supported by the service (for example, `tcp,udp`).

UID — User ID (real and effective) for the server process to be started.

`-` or `0` indicates default: same as the *albd_server* process.

GID — Group ID (real and effective) for the server process to be started.

`-` or `0` indicates default: same as the *albd_server* process.

Kind — Type of server: `unshared`, `reusable`, `shared`, `schedule`. ("schedule" means "schedulable".)

Control — A comma-separated, no-white-space list of server control parameters: (1) maximum number of servers allowed, (2) clients-per-server threshold (for schedulable servers only), (3) smoothed "busyness" threshold, at which a new instance of the server is created.

Program — Pathname of server executable. This may be a full pathname or a pathname relative to */usr/atria/etc*.

Arguments — (optional) Special arguments to include when starting a new instance of the server. Do not use `-` in this field; leave it blank instead.

Example:

```
390513  3  udp,tcp  -  -  shared         -       view_server
390514  3  udp      -  -  shared         -       vob_server
390515  3  tcp      -  -  reusable       -       db_server
390516  2  tcp      -  -  shared         -       promote_server
390518  2  tcp      -  -  schedule  5,0,5000000  vobrpc_server
```

OTHER CLEARCASE SERVERS

Several ClearCase servers do not run under *albd_server* control:

- The VOB database lock manager, *lockmgr*, is started by the same startup script as *albd_server*, at system startup time.
- The audited build executor, *abe*, is invoked by *clearmake*, as necessary, using the standard remote-shell facility.

OTHER ALBD_SERVER FUNCTIONS

In addition to its other duties, the *albd_server* performs these functions:

- On the network-wide *license server host*, *albd_server* fields license-verification requests from hosts throughout the network. In its role as the *license server* program, *albd_server* periodically consults the network's *license database file*, which must be availably locally as */usr/adm/atria/license.db*.

  See the *license.db* and *clearlicense* manual pages for more information.

- On the network-wide *registry server host*, *albd_server* fields requests for registry information from hosts throughout the network. In its role as the *registry server* program, *albd_server* periodically uses the network's *storage registry* files, which must be available locally in directory */usr/adm/atria/rgy*.

  See the *lsvob*, *lsview*, and *registry_ccase* manual pages for more information.

- During a distributed build, the *albd_server* process on a build server host fields load-balancing queries from the remote *clearmake* process. Its response to the query either allows the host to be used for build script execution, or to be bypassed. See the *bldserver.control* manual page for details.

FILES

/etc/services
/usr/atria/config/services/albd.conf
/usr/adm/atria/license.db

SEE ALSO

cleartool subcommands: register, mktag, lsview, lsvob
abe, clearlicense, cleartool, db_server, init_ccase, lockmgr, promote_server, view_server, vob_server
bldserver.control, license.db, registry_ccase

NAME bldhost – build hosts file / client-side control file for distributed build

SYNOPSIS

- Hosts to be considered for use in distributed build:

  hostname-1
  hostname-2
  ⋮

- Idleness threshold:

  –idle *percentage* [**%**]

- Control manner in which hosts are selected:

  –random

- Include-file facility:

  #include *pname*

DESCRIPTION

A *build hosts* file is a text file that specifies a list of *build server* hosts and, optionally, additional control information. This list is used by *clearmake* when you invoke it with the `–J` option, or equivalently, with the environment variable CLEARCASE_BLD_CONC set. Such a build is typically both *parallel* (multiple build scripts are executed concurrently) and *distributed* (build script execution is dispatched to one or more hosts in the network).

The build hosts file lists the hostnames, one per line, of machines that *clearmake* can use in a distributed build. *clearmake* dispatches build scripts to some or all these hosts using a load-balancing scheme (see below). The same host can be listed more than once; more work may be dispatched to such a host — presumably a multiprocessor or very fast processor that is capable of handling a heavy load. See also the description of the `–power` specification in the *bldserver.control* manual page.

Name of Build Hosts File

You can have several build hosts files, all of which must be stored in your home directory. Having several files is important for heterogeneous development environments — when building the HP-UX variant of a program, you probably don't want to dispatch build scripts to SunOS hosts. You might also use different build hosts files at different times — during the work day, overnight, over the weekend.

When it begins a distributed build, *clearmake* examines the environment and uses this file in your home directory:

```
.bldhost.$CLEARCASE_BLD_HOST_TYPE
```

(Your home directory is determined by examining the password database.)

LOAD BALANCING

ClearCase's load-balancing algorithm controls the way in which build scripts are dispatched to hosts. During the course of a distributed build, your *clearmake* process creates and updates a list of "qualified hosts", a subset of the hosts listed in the build hosts file. A host is "qualified" if all these criteria are met:

- The host is at least 50% idle (or your customized setting — see "Idleness Threshold" below).
- Your *clearmake* process meets the host's requirements, as specified in its *bldserver.control* file.
- An *abe* process can be started on the host.

Whenever it needs to dispatch a build script, *clearmake* spends some time updating its qualified hosts list, and then selects one of these hosts. If it cannot find any qualified host, it pauses and then updates the list again. *clearmake* keeps trying in this manner until it finds at least one qualified host with which to build.

The selected host is not necessarily the "best" one — for example, the one that is most idle at that particular moment.

Randomizing Host Selection

The default load-balancing algorithm tends to select hosts near the top of the list more often than those near the bottom of the list (subject to their availability). For more even-handed selection when the list of hosts exceeds 20 or so, include this line:

–random

Note that this also changes the effective location of any `#include` directives (see below).

Idleness Threshold

By default, your *clearmake* process will not dispatch a build script to a host unless it is at least 50% idle. You can adjust this *idleness threshold* with a line in the build hosts file:

–idle *percentage* [**%**]

percentage can be any integer from 0 to 100.

INCLUDE FILE FACILITY

A build hosts file can include the contents of one or more other build hosts files:

#include *pname*

If the included file has a `–random` directive, it applies just to that file's entries. `–idle` directives in an included file are ignored; you must set the idleness threshold at the top level.

Comment Lines

Any line that begins with `#` (except an `#include` line) is treated as a comment.

ORDER OF LINES

In any build hosts file, top-level or included, `–idle` and `–random` lines must precede all other lines.

EXAMPLES

- Build hosts file that uses a listed hosts only if it is at least 75% idle:

```
-idle 75
mercury
earth
mars
pluto
```

- Nesting of build hosts files:

```
-idle 30
einstein
bohr
fermi
#include /usr/local/lib/planet.hosts
```

SEE ALSO

"Parallel and Distributed Building" in the clearmake manual page
abe, bldserver.control, clearmake.options, makefile_ccase

NAME bldserver.control – server-side control file for distributed build

SYNOPSIS

- Load-Balancing Rule:

  [**–host** *host-list*] [**–user** *user-list*] [**–idle** *percentage* [**%**]]
  [**–time** *start-time*,*end-time* …]

- Comparative Power Specifier:

  –power *factor*

DESCRIPTION

Any ClearCase host can have a *build server control* file. This text file, */usr/adm/atria/config/bldserver.control*, specifies when, how, and by whom the host can be used as a build server in a distributed build.

During a distributed build, *clearmake* consults the user's *build hosts* file to determine which host(s) to use for executing build scripts. (See the *bldhost* manual page for details.) Before actually dispatching a build script, *clearmake* queries the *albd_server* process on the target build host, in essence asking "May I send you a build script?".

If the host's build server control file is missing or empty, no restrictions are placed on the use of the machine for distributed builds. The machine's *albd_server* will always send a "yes" response to the *clearmake* process controlling a distributed build.

If the host's build server control file is nonempty, *albd_server* examines the load-balancing rules one-by-one:

- If it finds a rule that "matches" the parameters of the current build, *albd_server* sends a "yes" response to the originating *clearmake*, which then uses a remote shell command to dispatch the build script.
- If no rule in the control file provides a match, *albd_server* sends a "no" response; the controlling *clearmake* proceeds to query another host.

For example, suppose this rule occurs in the control file:

```
-host jupiter -user *.dvt -time 21:00,07:30
```

This rule "matches" any build invoked on host *jupiter* between 9PM and 7:30AM, by a user whose principal group is *dvt*.

OPTIONS AND ARGUMENTS

Each of the following specifications is optional. A missing specification implies no restriction. The specifications are logically ANDed to form a test against the parameters of the current build.

–host *host-list*

One or more hosts, from which distributed build requests will be honored by this host. *host-list* is a comma-separated list (white space allowed); each item on the list is a hostname (as listed by *uname*(1)). The asterisk character (*) is a "wildcard" that matches all hostnames.

NOTE: Be sure to include the name of *this* host, if the command to perform a distributed build is (sometimes) entered here.

–user *user-list*

One or more users, whose builds will be permitted to use this host. *user-list* is a comma-separated list (white space allowed); each item on the list specifies a user — either by name or by number, either with a group qualifier or without. Examples of user specifications:

jones	User whose login name is *jones*
jones.dvt	User *jones*, but only if logged in with principal group *dvt*.
jones.*	Equivalent to specifying "jones" without any group qualifier.
566	User with user-ID 566

–idle *percentage* [**%**]

Allows use of this host only when its "idleness" is at least *percentage*, which must be an integer between 0 and 100, inclusive. Idleness is negatively correlated with the host's load factor, as shown by *uptime*(1); the approximate correspondence is:

Load	**Idle Percentage**
0.0	100
0.5	68
1.0	47
2.0	22
4.0	almost 0

–power *factor*

(must be specified alone, on a separate line) During the computation of the host's idleness, divides *factor* into the *percentage* specified with `–idle` (or into the system default). Thus, these two specifications are equivalent:

Spec 1	**Spec 2**
–idle 60	–idle 20
–power 3	

factor must be a non-negative floating-point number. This option allows you to model a "powerful" host — perhaps a multiprocessor — which is more capable of accepting work at a given idleness level. You might use `-power 3.0` or `-power 2.5` for a three-processor build server host. You can also model a relatively weak host, by assigning it a power value less than 1.0.

If a build server control file includes multiple `–power` lines, only the last one takes effect.

–time *start-time*,*end-time* …

Specifies one or more intervals during which the host will be available as a build server. *start-time* and *end-time* must be specified in 24-hour format:

hh:*mm* (*hh* = 0–23 ; *mm* = 0–59)

An interval can span midnight; for example, `17:00,8:00` specifies the interval from 5PM one evening to 8AM the following day.

EXAMPLES

- Allow builds by users *jackson* and *jones*, initiated from any host, if the host is at least 75% idle and the time is between 10PM and 6AM.

```
-host *  -user jackson,jones  -idle 75  -time 22:00,06:00
```

- Allow anyone to use this host for distributed builds between 7PM and 7AM.

```
-time 19:00,7:00
```

- Declare this host to be three times as powerful (able to handle distributed build requests) as a standard host.

```
-power 3.0
```

SEE ALSO

clearmake, abe, bldhost

NAME cc.icon, default.icon – file type to icon mapping rules (graphical interface)

SYNOPSIS

file-type [*file-type* …] **:** *icon-name* **;**

⋮

DESCRIPTION

An *icon file* contains an ordered set of rules that maps *file types* to names of *bitmap files*, which contain icon bitmaps.

In *xclearcase*, a file browser use a series of lookups to determine how to represent a file system object:

1. It searches one or more magic files to determine the list of *file types* for the file system object. (See the *cc.magic* manual page for details.)
2. It searches one or more icon files for a match with the first file type. Finding a match yields the name of a bitmap file. For example, this entry maps the file type "text_file" to the icon bitmap file name "text":

   ```
   text_file : -icon text ;
   ```

   The semicolon (;) character that terminates an icon rule must be preceded by white space.

   If no match can be found for the first file type, *xclearcase* searches the same set of icon files for a match with the second file type, and so on through the entire list of file types, if necessary. (If none of the file types produces a match in any icon file, an error occurs.)
3. Having determined the name of a bitmap file, *xclearcase* searches for an actual file in one or more directories containing bitmap files. (If it cannot locate a bitmap file with this name, an error occurs.)

   Bitmap file names must have a numeric suffix, indicating the size of the bitmap — for example, *text.60*. *xclearcase* selects that bitmap file whose name begins with the string specified by `-icon`, and whose size is appropriate for the current context.

Figure 12 illustrates this process.

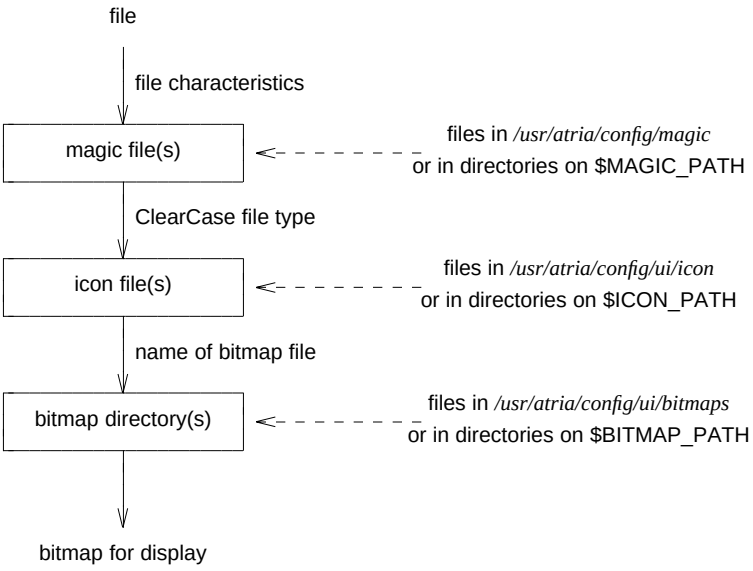


Figure 12. Bitmap Lookup Procedure

If the file system object is *selected*, this process includes an extra step: *xclearcase* tries to match a `-selected` icon rule for each relevant file type before accepting a bitmap specified by `-icon`. For example, the following rule specifies both generic and "when selected" icons for use with elements of type *text_file*:

```
text_file : -icon text -selected text_selected;
```

Selecting and deselecting a *text_file* object from a file browser toggles between the two icons.

Search Paths

ClearCase supports search paths both for icon files and for bitmap files:

- **Icon file search path** — If ICON_PATH is set in your environment (to a colon-separated list of directories), *xclearcase* searches files with a *.icon* suffix in these directories. In each directory, files are processed in alphabetical order. As soon as ClearCase finds a matching rule, the search ends; thus, if multiple rules match a file type, the *first* rule encountered wins.

  If ICON_PATH is not set, this default search path is used:

  home-directory`/.icon:${ATRIAHOME:-/usr/atria}/config/ui/icon`

- **Bitmap file search path** — If BITMAP_PATH is set in your environment (to a colon-separated list of directories), *xclearcase* searches for bitmap files with a *.60* suffix in these directories.

If BITMAP_PATH is not set, this default search path is used:

home-directory`/.bitmaps:${ATRIAHOME:-/usr/atria}/config/ui/bitmaps`

ClearCase also supports search paths for magic files; see the *cc.magic* manual page for details.

EXAMPLES

- For file type *c_source*, use the icon file named *c*. When a *c_source* element is selected, use the icon file *c_select*.

```
c_source : -icon c -selected c_select;
```

- For file type *postscript*, use the icon file named *ps*.

```
postscript : -icon ps ;
```

SEE ALSO

cc.magic

NAME cc.magic, default.magic – ClearCase file typing rules

SYNOPSIS

- File-typing rule:

  file-type-list : *selection-expression* **;**

- File type list:

  file-type [*file-type* …]

- Selection expression:

  selection-op [*arg(s)*] [*logical-op selection-op* [*arg(s)*]] …

DESCRIPTION

A *magic file* contains an ordered set of *file-typing rules*, which ClearCase uses to determine a list of *file types* for an existing file system object, or for one that is about to be created. A rule can use the object's name, its *file*(1) or *stat*(2) data, or its contents. File-typing involves searching one or more magic files for the first rule that matches a file system object; finding a match yields a single file type or an ordered list of file types; failing to find a match produces an error. ClearCase performs file-typing in these situations:

- When you create a new element with *mkelem*, but you do not specify an element type (with `–eltype`), the element's name is file–typed. (If you are converting a view–private file to an element with `mkelem -ci` or `mkelem -nco`, the file's contents are also used in the file-typing.) The resulting file type list is compared with the VOB's set of element types: the first file type that matches an element type is chosen as the element type; if no file type matches any existing element type, an error occurs:

  ```
  cleartool: Error: Can't pick element type from rules ...
  ```

- The ClearCase directory browsers have a graphical mode, in which each file system object is displayed as an icon. The icon is selected by first file-typing the object, then using one of its file types to select a bitmap from the ones listed in an *icon* file. (See the *cc.icon* manual page.)

Following are examples of file-typing rules:

```
directory : -stat d ;
c_source source text_file : -printable & -name "*.c" ;
sh_script script text_file : -printable & (-name ".profile" | -name "*.sh") ;
archive library file: !-printable & -name "*.a" ;
```

Search Path

ClearCase supports a search path for magic files. If MAGIC_PATH is set in your environment (to a colon-separated list of directories), *xclearcase* searches files with a *.magic* suffix in these directories. In each directory, files are processed in alphabetical order. As soon as ClearCase finds a matching rule, the search ends; thus, if multiple rules match a file type, the *first* rule encountered wins.

If MAGIC_PATH is not set, this default search path is used:

home-directory`/.magic:${ATRIAHOME:-/usr/atria}/config/magic`

FILE-TYPING RULES

Each file-typing rule has the following format:

file-type-list **:** *selection-expression* **;**

A single text line can contain multiple rules. Conversely, a single rule can span several lines — each intermediate line must end with a backslash (\) character. A line that begins with a pound-sign (#) character is a comment.

NOTE: The semicolon (;) character that terminates a rule must be separated from the preceding characters by white space.

FILE TYPE LIST

A *file-type-list* is an ordered list of one or more names, separated by white space. Only letters, digits, and underscore (_) characters are permitted in these names. Depending on the file-typing situation, each name should match either an element type defined in some VOB, or an icon name specified in an icon file. To avoid errors, always make the final name one of ClearCase's predefined element types: *file*, *text_file*, or *directory*. (These names are also included in the system-default icon file.)

Following are some *file-type-list* examples:

```
text_file
backup_dir directory
manual_page text_file
cplusplus_src src_file text_file
```

Here is a scenario that calls for a lengthy file type list:

Your host mounts several VOBs, in which different sets of element types are defined. Perhaps one VOB defines element type *bshell* for Bourne shell scripts, another VOB defines element type *shell_script* for all shell scripts, and yet another VOB does not define any special element type for scripts. Your file-typing rules must be appropriate for all the VOBs. For example:

```
bshell shell_script text_file : -name "*.sh" ;
shell_script text_file        : -name "*.csh" ;
```

With the above file-typing rules, *xclearcase* would use the file type *text_file* to select the same icon for all shell script files. A user who wished to distinguish Bourne shell scripts from C shell scripts might add a *cshell* file type, and create different bitmaps to correspond to the unique file types *bshell* and *cshell*.

Magic File:

```
bshell shell_script text_file : -name "*.sh" ;
cshell shell_script text_file : -name "*.csh" ;
```

Icon File:

```
bshell : bourne_shell_icon.bmap ;
cshell : C_shell_icon.bmap ;
```

SELECTION EXPRESSION

A *selection-expression* consists of one or more *selection operators* and their arguments, connected by *logical operators*. Examples:

```
-name "*.c"
-name "*.[ch]
```

```
-name "*.c" | -name "*.h"
-printable
!-printable
-stat d
```

Selection Operators and Arguments

Any abbreviation of a selection operator name is accepted. For example, you can abbreviate `–name` to `–n`, `–na`, or `–nam`.

All *string* arguments must be enclosed in double-quotes. Use `\"` to include a double-quote character in a string argument.

In the file system object already exists, any of the selection operators listed below can produce a match. If you are determining the file type for a non-existent object (for example, one that is about to be created), only the `–name` operator can produce a match.

–name *pattern*

Matches an object's simple file name (*leaf name*) against the specified pattern. *pattern* is a double-quoted string, and can include any ClearCase wildcard, except for *ellipsis* (`...`). See the *wildcards_ccase* manual page for a complete list.

–stat *stat_char*

Matches an object against the specified *stat*(2) file type. *stat_char* is a single character:

r	regular file
d	directory
c	character device
b	block device
f	FIFO (named pipe)
s	socket
l	symbolic link

NOTE: The selection expression `-stat l & -stat r` is TRUE for a symbolic link that points to a regular file. In general, however, testing for symbolic links is not particularly useful. *xclearcase* displays an icon for the object it finds at the end of a chain of symbolic links.

–magic *byte_offset, data_type, value*
–magic *byte_offset, string*

Matches an object against a *magic value* in the UNIX tradition: a number or string at a specified offset within the object's first physical block (512 bytes).

byte_offset The byte offset from the beginning of the file.

data_type The architecture-specific data format of the numeric *value* argument that follows:

byte	*value* is an 8-bit byte.
l_short	*value* is a little-endian 16-bit shortword.

b_short *value* is a big-endian 16-bit shortword.

l_long *value* is a little-endian 32-bit longword.

b_long *value* is a big-endian 32-bit longword.

value A numeric magic value, expressed as an integer in hex, octal, or decimal:

`0x` … a hexadecimal value

`0` … an octal value

… (any other form) a decimal value

string A non-numeric magic value, expressed as a double-quoted string.

–printable Matches an object if it is a printable file:

- Its first block must contain only characters evaluating to TRUE by the X/Open *isprint* and *isspace* routines.
- Its first block must have an average line length ≤ 256.

–token *string*
Matches an object if the specified double-quoted string occurs in its first physical block (512 bytes).

–file *string* Matches an object if the leading characters in its *file*(1) command output match the specified double-quoted string.

Logical Operators

File-typing rules can use the following logical operators, listed in decreasing order of precedence:

( )	parentheses for grouping
!	unary NOT
&	logical AND
&&	logical AND
|	logical OR
||	logical OR

NOTE: The effect of the unary NOT operator may depend on whether or not an object exists. It cannot produce a match if the selection operator is "inappropriate" — for example, attempting to *stat* a non-existent object:

```
! -stat f       (produces a match when file–typing the name of an existing directory)
! -stat f       (fails to produce a match when file–typing a name for which no object currently exists)
```

EXAMPLES

- Assign the file types *source_file* and *text_file* to files whose file name suffix is `.c` or `.h`.

```
source_file text_file : -name "*.c" | -name "*.h" ;
```

- Assign the file types *cplusplus_source* and *text_file* to printable files whose file name suffix is `.cxx` or `c++`.

```
cplusplus_source text_file : -printable & (-name "*.cxx" | -name "*.c++") ;
```

- Assign the file types *csh_script* and *text_file* to printable files that begin with the character string `#!`, and whose first block contains the string `csh`.

```
csh_script text_file : -printable & -magic 0,"#!" & -token "csh" ;
```

- Assign the file type *directory* to all directory objects.

```
directory : -stat d ;
```

- Assign the file types *cpio* and *file* to objects that the standard UNIX *file*(1) programs reports as "cpio archive".

```
cpio file : -file "cpio archive" ;
```

FILES

/usr/atria/config/magic/default.magic

SEE ALSO

cleartool subcommands: mkelem, mkeltype
cc.icon, wildcards_ccase, file(1), stat(2)

NAME clearaudit – non-clearmake build and shell command auditing facility

SYNOPSIS

clearaudit [*shell_cmd*]

DESCRIPTION

Runs an *audited shell* with the same view and working directory as the current process. *MVFS files* created within an audited shell (or any of its children) are *derived objects* (DOs). When it exits, an audited shell creates a *configuration record* (CR) and associates it with each of the newly-created DOs.

The CR and DOs produced by *clearaudit* are similar to those created by *clearmake*. They can be listed, compared, and deleted with the same *cleartool* commands used for other DOs (see below). They can be shared with other views through explicit *winkin* commands, but they cannot be winked-in by *clearmake*. They can be checked in as *DO versions*. See the *config_record* manual page for more on CRs produced by *clearaudit*.

clearaudit determines which program to run as follows:

- first choice — the value of environment variable CLEARAUDIT_SHELL, which must be the full pathname of a program …
- second choice — the value of environment variable SHELL, which must be the full pathname of a program, or …
- if neither of the above is set — the Bourne shell, */bin/sh*.

The process from which you invoke *clearaudit* must have a view context: *set view* or *working directory view*. In either case, the audited process is set to that view. An error occurs if the invoking process has no view context, or if its working directory view differs from its set view. (See the *pwv* manual page.)

By default, *clearaudit* creates temporary build audit files in directory */tmp*. You can set environment variable CLEARCASE_BLD_AUDIT_TMPDIR to specify an alternate location. All temporary files are deleted when *clearaudit* exits. CLEARCASE_BLD_AUDIT_TMPDIR must not name a directory under a VOB-tag; if it does, *clearaudit* displays an error message and exits.

Auditing Any Process

clearaudit can be used to document the work performed by any process. For example, you can use *clearaudit* to audit the creation of a *tar*(1) file, producing a configuration record that describes exactly which files and/or versions were written to tape. See the "EXAMPLES" section.

Auditing a non-ClearCase 'make'

You can also use *clearaudit* to produce derived objects and configuration records for software builds performed with another *make* program, such as UNIX *make*(1). Follow these guidelines:

- Set `SHELL=/usr/atria/bin/clearaudit` in the makefile.
- To prevent recursive invocation of *clearaudit*, set your process's CLEARAUDIT_SHELL environment variable to your normal shell (for example, */bin/sh*).
- If you want to produce a single CR for each target's build script, structure your makefiles so that each build script is a single shell command. Use continuation lines (\), as necessary.

Auditing a Shell Script

A shell script that begins with the following line is automatically executed in an audited shell:

```
#! /usr/atria/bin/clearaudit
```

Be sure that the process from which the script is invoked has CLEARAUDIT_SHELL set, as described above.

OPTIONS AND ARGUMENTS

shell_cmd One or more words, which are passed as arguments to $CLEARAUDIT_SHELL (or $SHELL, or */bin/sh*).

EXAMPLES

- Run program *myscr* in an audited C shell.

  ```
  % env SHELL=/bin/csh clearaudit myscr
  ```

- Run program *validation_suite* in an audited Korn shell.

  ```
  % setenv CLEARAUDIT_SHELL /bin/ksh
  % clearaudit validation_suite
  ```

- Following is a typical CR produced by *clearaudit*. It describes all files produced by a software build with UNIX *make*. View-private files are marked with time-modified stamps.

  ```
  Target ClearAudit_Shell built by block.user
  Host "starfield" running IRIX 4.0.1 (IP6)
  Reference Time 16-May-92.10:24:08, this audit started 16-May-92.10:24:08
  View was starfield:/usr/people/block/cc_views/view.bl62
  Initial working directory was /vobs/doc/reference_man/test
  ----------------------------
  MVFS objects:
  ----------------------------
  /vobs/doc/reference_man/test/hello@@16-May.10:25.16742
  /vobs/doc/reference_man/test/hello.c             <16-May-92.10:11:34>
  /vobs/doc/reference_man/test/hello.o@@16-May.10:25.16740
  /vobs/doc/reference_man/test/makefile            <16-May-92.10:23:57>
  ```

- Run a script that produces a tape backup in an audited shell; create an empty derived object (*tar_do*) whose CR will list all of the backed-up objects.

  ```
  audit_tar /dev/tape /usr/project
  ```

  Script *audit_tar*:

  ```
  #! /usr/atria/bin/clearaudit
  #
  echo "Audited tar backup of: $2"
  tar -cvf $1 $2

  echo "Creating derived object 'tar_do'"
  echo "" > ./tar_do

  exit 0
  ```

SEE ALSO

cleartool subcommands: catcr, diffcr, lsdo, pwv, rmdo, setview
clearmake, config_record, scrubber
make(1), sh(1), tar(1)

NAME clearbug – create problem report for Atria Customer Support

SYNOPSIS

clearbug [**–short** | **–s**]

DESCRIPTION

clearbug gathers information from your current processing context: date/time, version of operating system, versions of ClearCase tools, your UNIX and ClearCase contexts, system error logs, and so on. It sends this information to stdout, from which you can cut-and-paste it into a problem report for Atria Customer Support.

clearbug is self-documenting, displaying detailed instructions as it prompts you for information.

Send the problem report to your ClearCase support organization. The relevant information for the Atria Customer Support group follows.

- **By postal service** — Our mailing address is:

```
Customer Support Department
Atria Software, Inc.
24 Prime Park Way
Natick, MA 01760
```

- **By electronic mail** — Our Internet address is:

```
support@atria.com
```

OPTIONS AND ARGUMENTS

–short or **–s**

Suppresses the initial explanatory text, and proceeds straight to the first prompt.

NAME clearcvt_ccase – copy ClearCase data to a different VOB

SYNOPSIS

clearcvt_ccase [**−I** *date-time* | **−s** *date-time*] [**−r**] [**−n**] [**−o** *script-dir-pname*]
[**−p** *file-pname*] [**−e** *file-pname*] [*source-name …*]

DESCRIPTION

The *clearcvt_ccase* utility plays a central role in cross-VOB maintenance:

- moving an element from one VOB to another
- moving a directory element, along with all the elements and links cataloged within it, from one VOB to another
- moving an entire hierarchy of directory elements, file elements, and VOB links from one VOB to another
- splitting a VOB into two or more VOBs

In all these tasks, *clearcvt_ccase* does not itself delete any data from the original VOB — it creates conversion scripts that *copy* data to another location, (presumably) in a different VOB. To complete a move or a split, you must "get rid" of the original elements:

- Using *rmelem* on the original data actually removes the data from the VOB. This reclaims disk space — likely to be the motivation for performing the maintenance. But once elements are removed, you can no longer regenerate old source configurations that included them, and you cannot rebuild the corresponding software releases.
- Using *rmname* on the original data makes it disappear from the directory hierarchy, but does not reclaim any disk space. This strategy allows old source configurations to be regenerated, and old releases to be rebuilt.

What Gets Converted / What Does Not

This section describes in detail what aspects of ClearCase objects are converted by *clearcvt_ccase*. Note that much, but not all of the associated *meta-data* is converted.

Directory Elements. *clearcvt_ccase* does not convert entire directory elements. Rather, it converts the contents of (the elements and links cataloged in) one particular version of a directory — the version selected by the current view. Thus, even converting an entire VOB might miss some of the VOB's elements — the ones that are not included in the VOB namespace, as it is currently configured by the view.

File Elements. For each file element, *clearcvt_ccase* converts the element itself, along with some or all of its versions. Command options control which versions are converted.

If the element has a user-defined element type, an error occurs if the conversion script is executed in a VOB in which that element type is not defined. (No effort is made to verify that the element type is defined the same way in both VOBs.) Any attributes attached to an element object itself are not converted.

Versions of File Elements. For each file element version it processes, *clearcvt_ccase* converts the version's contents, its version labels, and its attributes. Checked-out versions are never converted; the conversion script issues a warning message and otherwise ignores a checked-out version.

Information Not Converted. *clearcvt_ccase* does not convert *hyperlinks* or *triggers*. Exception: hyperlinks that represent merges (those of type *Merge*) are converted.

The contents of a VOB's *lost+found* are converted only if you make it the current directory before entering the *clearcvt_ccase* command.

CONVERSION PROCESS

The conversion process consists of two stages: export and import.

Export Stage

Any user can enter the 'clearcvt_ccase' command that implements the export stage.

The export stage takes place in the VOB where the data to be moved resides. *clearcvt_ccase* creates a set of *conversion scripts* (Bourne shell scripts), containing commands to create elements, branches, versions, and associated meta-data.

In the import stage, you will use the *master conversion script* to invoke all the individual scripts. If any of the files to be converted reside below (rather than *in*) the current working directory, the master conversion script also includes commands to create corresponding directory element(s).

For each element it processes, *clearcvt_ccase* extracts versions and writes a conversion script that either

- creates a new element with the same versions as the original, or
- checks out an existing element (optionally, on a branch) and checks in a new version for each original version that has not already been converted.

You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

Import Stage

Only a VOB's owner or the root user can run the conversion scripts that implement the import stage (unless 'clearcvt_ccase –n' option was used to create the scripts).

The import stage takes place within an existing VOB, in a shell whose view uses the ClearCase default config spec. In this stage, you execute the master conversion script created by *clearcvt_ccase*, populating the VOB with new elements and versions. These changes to the VOB are documented with event records:

- Each time an individual script creates a new file element, an `import file element` event record is stored in the VOB database, along with the standard `create element` event record. It is associated with the parent directory element, not with the new file element itself. Heuristic: the "import" event record is created only if the object is more than 24 hours old.

- Each time a script creates a new version, the standard `create version` event record is annotated with the comment from the original version.
- The `import file element` event record is always stamped with the current time. The `create version` and `create element` event records are timestamped according to the original data (unless `clearcvt_ccase -n` was used to create the conversion scripts).
- The event record for the creation of a branch gets the same timestamp as the branch's first version.
- The event record for the attaching of an attribute gets the same timestamp as the associated version.
- Event records for the creation of directory elements and type objects are stamped with the current time.

Incremental Conversion and Restartability. The conversion scripts created by *clearcvt_ccase* can skip certain versions, or entire elements, during the import stage. This capability enables valuable features:

- You can convert one element to another in several conversion passes. You might use incremental conversion for time-budgeting reasons (there are too many versions to convert all at once), or because the original element is still under development.
- If execution of the master conversion script terminates prematurely for any reason, you can restart it; it will skip versions it has already converted, effectively resuming where it left off.

  CAUTION: Conversion scripts created with `clearcvt_ccase -n` are not restartable.

An import script decides whether to skip an element using a heuristic: if the target branch of the target element already has a version created at the same time (or later than) the most recent version in the source element, the source element is skipped. You can suppress this element-skipping algorithm by running the conversion script in an environment where CVT_UPDATE is set to any non-null string. This forces the conversion script to consider converting each source version.

For each source version, an import script will decline to create a corresponding version if it already exists on the target branch — that is, if it has the same time-modified stamp (or a more recent one). But even when it bypasses version-creation, the import script still updates the new version's meta-data (for example, version labels) using information from the source version.

Background Execution of the Import Stage. You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

Special Characters in File Names

clearcvt_ccase can handle file names that include some, but not all, of the characters that are special to the UNIX shells. Conversion fails for any file name that includes any of these characters:

```
‘   ’   "   <Space>   <Tab>   [   ]   ?   *
```

For example:

Succeeds	**Fails**
foo&bar	foo bar
$MY_LIB	yellow'sunset
yellow(Y)	dog*bert

OPTIONS AND ARGUMENTS

Storage Location of Conversion Scripts. *Default:* Subdirectory *cvt_dir* is created in the current working directory, and all the conversion scripts are stored there. An error occurs if *./cvt_dir* already exists.

–o *script-dir-pname*
Creates the specified directory and stores the conversion scripts there. An error occurs if the directory already exists.

Handling of Directory Arguments. The *source-name* argument specifies the version of a directory element currently selected by your view. By default: (1) a directory element will be created in the target VOB for *source-name* itself; (2) for each file element cataloged in the currently-selected version of *source-name*, an individual conversion script is created (3) for each directory element cataloged in the currently-selected version of *source-name*, an empty directory element will be created in the target VOB — the contents of such (sub)directory elements are ignored.

–r *clearcvt_ccase* descends recursively into all *source-name* arguments that are directories. The recursive descent involves only the currently-selected version of each directory element.

Preprocessing / Postprocessing. *Default:* The import stage consists solely of the execution of the ClearCase-generated commands in the conversion scripts.

–p *file-pname*
Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed before any files are converted.

–e *file-pname*
Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed after all files are converted.

Transcription of History Information. *Default: clearcvt_ccase* extracts historical information from each element and places it in the element's conversion script. Thus, new versions will have the same *stat*(2) information — user, group, and time-modified stamp — as the original versions. The `create version` and `create element` event records created by the conversion script also get this original information.

–n Event records and *stat* information for new elements and versions reflect the "who" and "when" of the execution of the conversion script, not the original data.

CAUTION: Using this option creates conversion scripts that are not restartable.

Selective Conversion of Files. *Default: clearcvt_ccase* converts entire elements.

–I *date-time* Converts "important" versions only, but includes *all* versions created since the specified time. Important versions are those with labels, those at which branches were created, and those at the ends of branches. The time is specified as follows:

date-time	:=	*date***.***time* | *date* | *time* | **now**
date	:=	*day-of-week* | *long-date*
day-of-week	:=	**today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date	:=	*d*[*d*]–*month*[–[*yy*]*yy*]

month := **January** | ... | **December** | **Jan** | ... | **Dec**
time := *h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

–s *date-time* *This option is designed for regular, incremental updating of an element from another one that is still under development. Be sure to specify a 'date-time' that covers the entire period since the preceding update. In other situations, it is probably better to use '–I' instead of '–s'.*

Only versions created since the specified time are processed. Exception: a branch created at an "old" version is converted if one or more "new" versions exist on the branch, or if environment variable CVT_UPDATE is set to a non-null value.

NOTE: In an incremental updating situation, removal of a label or branch from an imported version is not propagated to the target element.

Specifying Files to be Converted. *Default:* The current working directory (equivalent to specifying "." as the *source-name* argument). Each element in the current working directory will be recreated in the target VOB; a directory element will be created in the target VOB for each subdirectory of the current working directory.

source-name …

One or more pathnames, specifying elements and/or directory versions:

- For each specified element, the conversion script will recreate some or all of its versions.
- For each specified directory version, conversion scripts are created for all the elements it catalogs. Commands are placed in the master conversion script to create a directory element for the specified directory itself, and for its subdirectories.

Each *source-name* must be a simple file or directory name. This enables the conversion scripts to reliably access the source data when they are executed. Specifying the parent directory (..) causes an error, as does any pathname that includes a slash (/) character.

Thus, before entering this command, you should change to the directory where (or under which) the elements to be converted reside.

EXAMPLES

- Create conversion scripts for the entire tree under directory element *src*, converting only versions created since the beginning of 1993.

```
% clearcvt_ccase –r –s 1–Jan–1993 src
```

- Create conversion scripts for the elements in the current working directory, but not in any subdirectories; store the scripts in a new subdirectory *newcvt*, of your home directory.

```
% clearcvt_ccase –o ~/newcvt .
```

SEE ALSO

cleartool subcommands: chtype, protect, rntype, setview
clearcvt_dsee, clearcvt_rcs, clearcvt_sccs, clearcvt_unix, events_ccase
rcs(1), rsh(1) or remsh(1), sccs(1)

NAME clearcvt_dsee – convert DSEE elements to ClearCase elements

SYNOPSIS

clearcvt_dsee [**−I** *date-time* | **−s** *date-time*] [**−r**] [**−n**] [**−o** *script-dir-pname*]
[**−p** *file-pname*] [**−e** *file-pname*] [**−T** *translation-file*]
[*source-name* …]

DESCRIPTION

clearcvt_dsee converts data created by versions 3.3.*n* and 4.*n* of the *Domain Software Engineering Environment* (*DSEE*), running under Domain/OS Version SR10.*n*. *clearcvt_dsee* is a Domain/OS executable; it cannot be run on UNIX systems.

The following DSEE constructs are converted directly to the like-named ClearCase constructs: elements, branches, obsolete branches, versions, and version labels.

CONVERSION PROCESS

The conversion process consists of two stages: export and import.

Export Stage

Any user can enter the 'clearcvt_dsee' command that implements the export stage.

The export stage takes place in the VOB where the data to be moved resides. *clearcvt_dsee* creates a set of *conversion scripts* (Bourne shell scripts), containing commands to create elements, branches, versions, and associated meta-data.

In the import stage, you will use the *master conversion script* to invoke all the individual scripts. If any of the files to be converted reside below (rather than *in*) the current working directory, the master conversion script also includes commands to create corresponding directory element(s).

For each DSEE element it processes, *clearcvt_dsee* extracts versions and writes a conversion script that either

- creates a new element with the same versions as the original, or
- checks out an existing element (optionally, on a branch) and checks in a new version for each original version that has not already been converted.

You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

Import Stage

Only a VOB's owner or the root user can run the conversion scripts that implement the import stage (unless 'clearcvt_dsee –n' option was used to create the scripts).

The import stage takes place within an existing VOB, in a shell whose view uses the ClearCase default config spec. In this stage, you execute the master conversion script created by *clearcvt_dsee*, populating the VOB with new elements and versions. These changes to the VOB are documented with event records:

- Each time an individual script creates a new file element, an `import file element` event record is stored in the VOB database, along with the standard `create element` event record. It is associated with the parent directory element, not with the new file element itself. Heuristic: the "import" event record is created only if the object is more than 24 hours old.
- Each time a script creates a new version, the standard `create version` event record is annotated with the comment from the original version.
- The `import file element` event record is always stamped with the current time. The `create version` and `create element` event records are timestamped according to the original data (unless `clearcvt_dsee -n` was used to create the conversion scripts).
- The event record for the creation of a branch gets the same timestamp as the branch's first version.
- The event record for the attaching of an attribute gets the same timestamp as the associated version.
- Event records for the creation of directory elements and type objects are stamped with the current time.

Incremental Conversion and Restartability. The conversion scripts created by *clearcvt_dsee* can skip certain DSEE versions, or entire DSEE elements, during the import stage. This capability enables valuable features:

- You can convert a DSEE element to a ClearCase element in several conversion passes. You might use incremental conversion for time-budgeting reasons (there are too many versions to convert all at once), or because the DSEE element is still under development.
- If execution of the master conversion script terminates prematurely for any reason, you can restart it; it will skip versions it has already converted, effectively resuming where it left off.

  CAUTION: Conversion scripts created with `clearcvt_dsee -n` are not restartable.

An import script decides whether to skip an entire DSEE element using a heuristic: if the target branch of the ClearCase element already has a version created at the same time (or later than) the most recent version in the source DSEE element, the DSEE element is skipped. You can suppress this element-skipping algorithm by running the conversion script in an environment where CVT_UPDATE is set to any non-null string. This forces the conversion script to consider converting each DSEE version.

For each DSEE version, an import script will decline to create a corresponding version if it already exists on the target ClearCase branch — that is, if it has the same time-modified stamp (or a more recent one). But even when it bypasses version-creation, the import script still updates the new version's meta-data (for example, version labels) using information from the DSEE version.

Background Execution of the Import Stage. You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

SPECIAL CHARACTERS IN FILE NAMES

clearcvt_dsee can handle file names that include some, but not all, of the characters that are special to the UNIX shells. Conversion fails for any file name that includes any of these characters:

`‘ ’ " <Space> <Tab> [ ] ? *`

For example:

Succeeds	**Fails**
foo&bar	foo bar
$DSEE_LIB	yellow’sunset
yellow(Y)	dog*bert

TRANSLATION OF BRANCHES AND VERSION LABELS

DSEE allows branches and version labels to have the same names; but a label type cannot have the same name as a branch type (within the same VOB). If *clearcvt_dsee* encounters a label-branch naming conflict, it renames one of them. For example, the DSEE version label `rel2` might become the ClearCase label type `rel2___1`. Such renaming can introduce inconsistencies over multiple runs of *clearcvt_dsee*. The same label might be renamed during conversion of some DSEE libraries, but remain unchanged during conversion of others. You can enforce consistency by using the same *converter translation file* in multiple invocations of *clearcvt_dsee*. If you name such a file, using the **–T** option, *clearcvt_dsee* uses it to:

- Look up each DSEE label or branch to see how to translate it to a ClearCase label type or branch type. If a match is found, the DSEE label or branch is translated the same way.
- Record each translation of a new DSEE label or branch, for use in future lookups.

The first time you use *clearcvt_dsee*, use `–T` to create a new translation file. On subsequent invocations of *clearcvt_dsee*, use `–T` again, specifying the same translation file, for consistent name translation.

Each line of the translation file has three fields, indicating one DSEE-to-ClearCase translation:

1. the keyword `label` or `branch`
2. a DSEE label or branch
3. the corresponding ClearCase label type or branch type

REMOTE ACCESS TO DSEE LIBRARIES

The conversion scripts created by *clearcvt_dsee* contain commands of this form:

rcp ${CLEARCASE_REMOTE_USER}*hostname***://***hostname***/***path_to_dseelib***/***file*

In such commands, *hostname* specifies the host on which the DSEE source library resides. Make sure that such *rcp*(1) commands will succeed before running the conversion scripts. Setting the environment variable CLEARCASE_REMOTE_USER may help at sites where Domain/OS hosts have dissimilar password files. For example, setting this EV to `jones@` effectively converts target hostname *jupiter* to *jones@jupiter*.

CAUTION: The *rcp* command uses the *size_cache* value for each DSEE version. This value may be incorrect for versions created with releases prior to DSEE 3.3, causing problems: if the value is too small, the file is truncated during the copy; if the value is too big, the last blocks of the file are copied multiple times. Before converting a DSEE library, be sure that all of its elements' *size_cache* values are correct. To fix an incorrect value, use the DSEE command `recover library -add_size_cache -force`. (This can be quite time-consuming!)

OPTIONS AND ARGUMENTS

Storage Location of Conversion Scripts. *Default:* Subdirectory *cvt_dir* is created in the current working directory, and all the conversion scripts are stored there. An error occurs if *./cvt_dir* already exists.

–o *script-dir-pname*
: Creates the specified directory and stores the conversion scripts there. An error occurs if the directory already exists.

Handling of Directory Arguments. The *source-name* argument specifies the version of a directory element currently selected by your view. By default: (1) a directory element will be created in the target VOB for *source-name* itself; (2) for each file element cataloged in the currently-selected version of *source-name*, an individual conversion script is created (3) for each directory element cataloged in the currently-selected version of *source-name*, an empty directory element will be created in the target VOB — the contents of such (sub)directory elements are ignored.

–r
: *clearcvt_dsee* descends recursively into all *source-name* arguments that are directories. The recursive descent involves only the currently-selected version of each directory element.

Preprocessing / Postprocessing. *Default:* The import stage consists solely of the execution of the ClearCase-generated commands in the conversion scripts.

–p *file-pname*
: Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed before any files are converted.

–e *file-pname*
: Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed after all files are converted.

Transcription of History Information. *Default: clearcvt_dsee* extracts historical information from each DSEE element and places it in the element's conversion script. Thus, new versions will have the same *stat*(2) information — user, group, and time–modified stamp — as the original versions. The `create version` and `create element` event records created by the conversion script also get this original information.

–n
: Event records and *stat* information for new elements and versions reflect the "who" and "when" of the execution of the conversion script, not the original data.

  CAUTION: Using this option creates conversion scripts that are not restartable.

Handling of Branches and Version Labels. *Default:* As described above in the section "Translation of Branches and Version Labels", *clearcvt_dsee* may automatically rename a branch or label type on import to avoid naming conflicts.

–T *translation-file*
Uses the specified converter translation file to control the mapping from DSEE branches and version labels to ClearCase branch and label types.

Selective Conversion of Files. *Default: clearcvt_dsee* converts entire DSEE elements.

–I *date-time* Converts "important" versions only, but includes *all* versions created since the specified time. Important versions are those with labels, those at which branches were created, and those at the ends of branches. The time is specified as follows:

date-time	:=	*date***.***time* | *date* | *time* | **now**
date	:=	*day-of-week* | *long-date*
day-of-week	:=	**today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date	:=	*d*[*d*]–*month*[–[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

–s *date-time* *This option is designed for regular, incremental updating of a ClearCase element from a DSEE element that is still under development. Be sure to specify a 'date-time' that covers the entire period since the preceding update. In other situations, it is probably better to use '–I' instead of '–s'.*

Only DSEE versions created since the specified time are processed. Exception: a branch created at an "old" version is converted if one or more "new" versions exist on the branch, or if environment variable CVT_UPDATE is set to a non-null value.

NOTE: In an incremental updating situation, removal of a label or branch from a DSEE version is not propagated to the ClearCase element.

Specifying Files to be Converted. *Default:* The current working directory (equivalent to specifying "." as the *source-name* argument). Each element in the current working directory will be recreated in the target VOB; a directory element will be created in the target VOB for each subdirectory of the current working directory.

source-name …
One or more pathnames, specifying elements and/or directory versions:

- For each specified element, the conversion script will recreate some or all of its versions.
- For each specified directory version, conversion scripts are created for all the elements it catalogs. Commands are placed in the master conversion script to create a directory element for the specified directory itself, and for its subdirectories.

Each *source-name* must be a simple file or directory name. This enables the conversion scripts to reliably access the source data when they are executed. Specifying the parent directory (..) causes an error, as does any pathname that includes a slash (/) character.

Thus, before entering this command, you should change to the directory where (or under which) the elements to be converted reside.

EXAMPLES

- Create conversion scripts for a single DSEE element.

```
% clearcvt_dsee lib.c
```

- Convert three DSEE elements in the current working directory to elements in VOB directory */usr/src/project/include*; store the conversion scripts in directory *cvt_include*.

```
% clearcvt_dsee –o cvt_include lib{1,2,3}.h
% set S = ‘pwd‘
% cd /usr/src/project/include
% $S/cvt_include/cvt_script
```

SEE ALSO

cleartool subcommands: chtype, protect, setview
clearcvt_ccase, clearcvt_rcs, clearcvt_sccs, clearcvt_unix, events_ccase
rcs(1), rsh(1) or remsh(1), sccs(1)

NAME clearcvt_rcs – convert RCS files to ClearCase elements

SYNOPSIS

clearcvt_rcs [**–I** *date-time* | **–s** *date-time*] [**–r**] [**–n**] [**–o** *script-dir-pname*]
[**–p** *file-pname*] [**–e** *file-pname*] [**–V**] [**–T** *translation-file*] [**–S**]
[*source-name …*]

DESCRIPTION

Any user can create conversion scripts with this utility. If the '–n' option is not used to create the scripts, only the VOB owner or the 'root' user can run them. We recommend that the VOB owner run the scripts, to minimize the likelihood of permissions problems in multiple-host conversion situations.

clearcvt_rcs converts Revision Control System (RCS) files into ClearCase *elements* and *versions*. The source data for a conversion can range from a single file an entire directory tree.

clearcvt_rcs ignores most information in RCS files that is not related to version-tree structure. It converts each *RCS symbol*, which names a revision or branch, into the appropriate ClearCase construct: version label or branch. (You can specify a *translation file* to control this conversion, enforcing consistency over multiple invocations of *clearcvt_rcs*.) You can use the `–S` and `–V` options to preserve RCS state attributes and RCS revisions numbers as attributes of the corresponding ClearCase versions.

RCS Files, Working Files, and Locks

clearcvt_rcs works directly with the structured *RCS files*, which have the *,v* file name suffix. It does not convert the *working files* created with `co` and `co -l` commands; it merely issues warning messages indicating which files are checked-out. Be sure to check in working files with the *ci* command before running the converter.

Other than the issuing of warning messages for checked-out files *clearcvt_rcs* ignores all RCS locks.

RCS files can (but need not) be "buried" in *RCS* subdirectories; if they are, the subdirectory level is collapsed in the conversion process — for example, RCS file *./proj/RCS/main.c,v* becomes element *./proj/main.c*.

CONVERSION PROCESS

The conversion process consists of two stages: export and import.

Export Stage

Any user can enter the 'clearcvt_rcs' command that implements the export stage.

The export stage takes place in the area where the RCS files reside. *clearcvt_rcs* creates a set of *conversion scripts* (Bourne shell scripts), containing commands to create elements, branches, and versions.

In the import stage, you will use the *master conversion script* to invoke all the individual scripts. If any of the files to be converted reside below (rather than *in*) the current working directory, the master conversion script also includes commands to create corresponding directory element(s).

For each RCS file it processes, *clearcvt_rcs* extracts versions and writes a conversion script that either

- creates a new element and checks in a version for each RCS revision, or
- checks out an existing element (optionally, on a branch), and checks in a new version for each RCS revision that has not already been converted

clearcvt_rcs automatically "chases" symbolic links it encounters during the export stage.

Import Stage

Only a VOB's owner or the root user can run the conversion scripts that implement the import stage (unless 'clearcvt_rcs –n' option was used to create the scripts).

The import stage takes place within an existing VOB, in a shell whose view uses the ClearCase default config spec. In this stage, you execute the master conversion script created by *clearcvt_rcs*, populating the VOB with new elements and versions. These changes to the VOB are documented with event records:

- Each time an individual script creates a new file element, an `import file element` event record is stored in the VOB database, along with the standard `create element` event record. It is associated with the parent directory element, not with the new file element itself. Heuristic: the "import" event record is created only if the object is more than 24 hours old.
- Each time a script creates a new version, the standard `create version` event record is annotated with the comment from the RCS revision.
- The `import file element` event record is always stamped with the current time. The `create version` and `create element` event records are timestamped according to the original RCS data (unless `clearcvt_rcs -n` was used to create the conversion scripts).
- The event record for the creation of a branch gets the same timestamp as the branch's first version.
- The event record for the attaching of an attribute gets the same timestamp as the associated version.
- Event records for the creation of directory elements and type objects are stamped with the current time.

Incremental Conversion and Restartability. The conversion scripts created by *clearcvt_rcs* can skip certain RCS revisions, or entire RCS files, during the import stage. This capability enables valuable features:

- You can convert an RCS file to a ClearCase element in several conversion passes. You might use incremental conversion for time-budgeting reasons (there are too many revisions to convert all at once), or because the RCS file is still under development.
- If execution of the master conversion script terminates prematurely for any reason, you can restart it; it will skip revisions it has already converted, effectively resuming where it left off.

  CAUTION: Conversion scripts created with `clearcvt_rcs -n` are not restartable.

An import script decides whether to skip an entire RCS file using a heuristic: if the target branch of the ClearCase element already has a version created at the same time (or later than) the most recent revision in the source RCS file, the element is skipped. You can suppress this element-skipping algorithm by running the conversion script in an environment where CVT_UPDATE is set to any non-null string. This forces the conversion script to consider converting each RCS revision.

For each RCS revision, an import script will decline to create a corresponding version if it already exists on the target ClearCase branch — that is, if it has the same time-modified stamp (or a more recent one). But even when it bypasses version-creation, the import script still updates the version's meta-data (for example, version labels) using information from the RCS revision.

Background Execution of the Import Stage. You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

SPECIAL CHARACTERS IN FILE NAMES

clearcvt_rcs can handle file names that include some, but not all, of the characters that are special to the UNIX shells. Conversion fails for any file name that includes any of these characters:

```
‘   ’   "   <Space>   <Tab>   [   ]   ?   *
```

For example:

Succeeds	**Fails**
foo&bar	foo bar
$RCS_LIB	yellow'sunset
yellow(Y)	dog*bert

HANDLING OF RCS SYMBOLS

An *RCS symbol* is a (presumably) mnemonic name for a particular revision or branch of an RCS file. *clearcvt_rcs* translates the symbols to version labels and branch names (more precisely, to names of label types and branch types).

- **Translation to version labels** — Suppose an RCS symbol, *RLS_1.3*, names a revision, *3.5*. In the conversion script for that revision, *clearcvt_rcs* places commands to create label type *RLS_1.3*, and to assign a label of that type to the ClearCase version created from the RCS revision.
- **Translation to branch names** — Suppose an RCS symbol, *rls_1.3_fixes*, names a branch, *3.5.1*. *clearcvt_rcs* outputs conversion-script commands to create branch type *rls_1.3_fixes*, and to create a branch of that type at the ClearCase version created from RCS revision *3.5*.

Single-digit symbols that name RCS branches are not converted, since there is no ClearCase concept of a "subbranch" of the *main* branch. If an RCS symbol includes characters that are not valid in names of label types or branch types, the offending characters are replaced by dot (`.`) characters. For example, the RCS symbol `C++` would have `C..` as its ClearCase translation.

A label type cannot have the same name as a branch type (within the same VOB). If the same RCS symbol names both a revision and a branch — not necessarily in the same RCS file — *clearcvt_rcs* renames one of them. For example, after converting a symbol *FX354*, which names a branch, it might encounter the same symbol as the name of a revision in another RCS file. In this case, it would create and use label type *FX354___1* in the ClearCase element.

Converter Translation File

This renaming of RCS symbols can introduce inconsistencies over multiple runs of *clearcvt_rcs*. The same symbol might be renamed during conversion of some RCS files, but remain unchanged during conversion of other files. You can enforce consistency by using the same *converter translation file* in multiple

invocations of *clearcvt_rcs*. If you name such a file, using the `-T` option, *clearcvt_rcs* uses it to:

- Look up each RCS symbol to see how to translate it to a ClearCase label type or branch type. If a match is found, the symbol is translated the same way.
- Record each translation of a new RCS symbol, for use in future lookups.

The first time you use *clearcvt_rcs*, use `-T` to create a new translation file. On subsequent invocations of *clearcvt_rcs*, use `-T` again, specifying the same translation file, for consistent name translation.

Each line of the translation file has three fields, indicating one RCS-to-ClearCase translation:

1. the keyword `label` or `branch`
2. an RCS symbol
3. the corresponding ClearCase label type or branch type

VERSION TREE STRUCTURE AFTER CONVERSION

Revisions on the main branch of an RCS file have two-digit identifiers (for example, `1.2`). These revisions become versions on the *main* branch of the ClearCase element, as illustrated in Figure 13.

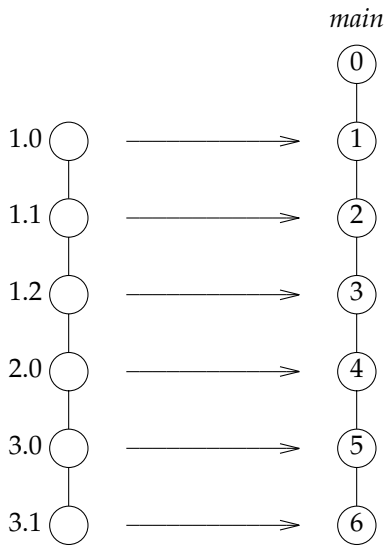


Figure 13. Conversion of RCS Revisions

Note that the "major revision" substructure in the RCS revision tree is lost in the translation — all the RCS revisions become versions on the *main* branch. (But you can use the `-V` option to preserve this information in the form of attributes attached to the versions.)

Revisions on subbranches of an RCS file have four-digit identifiers (for example, `1.2.1.5`). These revisions become versions on subbranches of the ClearCase element, as illustrated in Figure 14.

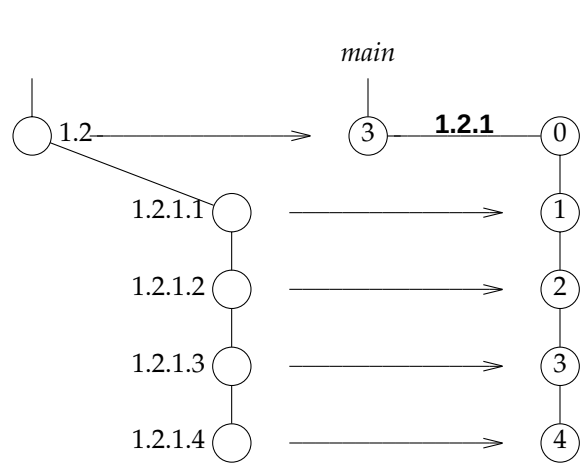


Figure 14. Conversion of RCS Subbranches

The conversion script creates branch types with three-digit names (`1.2.1` in the example above). Thus, RCS revision 1.2.1.3 becomes version *3* on branch *1.2.1.*

OPTIONS AND ARGUMENTS

Storage Location of Conversion Scripts. *Default:* Subdirectory *cvt_dir* is created in the current working directory, and all the conversion scripts are stored there. An error occurs if *./cvt_dir* already exists.

−o *script-dir-pname*
: Creates the specified directory and stores the conversion scripts there. An error occurs if the directory already exists.

Handling of Directory Arguments. *Default:* If a UNIX directory is specified as a *source-name* argument: (1) the RCS files in that directory are converted; (2) a directory element is created for *source-name* and for each of its subdirectories; (3) the contents of the subdirectories are ignored.

−r
: *clearcvt_rcs* descends recursively into all *source-name* arguments that are directories.

Preprocessing / Postprocessing. *Default:* The import stage consists solely of the execution of the ClearCase-generated commands in the conversion scripts.

−p *file-pname*
: Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed before any files are converted.

–e *file-pname*
Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed after all files are converted.

Transcription of History Information. *Default: clearcvt_rcs* extracts historical information from each RCS file and places it in the conversion script. (It also makes some system calls, such as *getgrgid*(3), to supplement this information.) Thus, the corresponding ClearCase version will have the same *stat*(2) information — user, group, and time–modified stamp — as the original RCS revision. The `create version` and `create element` event records created by the conversion script also get this original information.

–n Event records and *stat* information for new elements and versions reflect the "who" and "when" of the execution of the conversion script, not the original data.

CAUTION: Using this option creates conversion scripts that are not restartable.

Handling of Branches and Labels. *Default:* As it converts RCS symbols to version labels and branch names, *clearcvt_rcs* does not enforce consistency over multiple source files. This means, for example, that the same symbol in two RCS files might be converted to different version labels in the ClearCase elements.

–T *translation-file*
Uses the specified converter translation file to control and record the conversion of RCS symbols to version labels and branch names. See the "Converter Translation File" section above.

Selective Conversion of Files. *Default: clearcvt_rcs* converts all RCS revisions it finds.

–I *date-time* Converts "important" revisions only, but includes *all* revisions created since the specified time. Important revisions are those with labels, those at which branches were created, and those at the ends of branches. The time is specified as follows:

date-time	:=	*date***.***time* | *date* | *time* | **now**
date	:=	*day-of-week* | *long-date*
day-of-week	:=	**today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date	:=	*d*[*d*]–*month*[–[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

–s *date-time* *This option is designed for regular, incremental updating of an element from an RCS file that is still under development. Be sure to specify a 'date-time' that covers the entire period since the preceding update. In other situations, it is probably better to use '–I' instead of '–s'.*

Only RCS revisions created since the specified time are processed. Exceptions: a label attached to an "old" revision is converted; a branch created at an "old" revision is converted if one or more "new" revisions exist on the branch, or if environment variable CVT_UPDATE is set to a non-null value.

NOTE: In an incremental updating situation, removal of a label or branch from an RCS revision is not propagated to the ClearCase element.

Specifying Files to be Converted. *Default:* The current working directory (equivalent to specifying "." as the *source-name* argument). Each RCS file in the current working directory will be converted; a directory element will be created for each subdirectory of the current working directory (except one named *RCS*).

source-name …

One or more pathnames, specifying RCS files and/or directories:

- For each specified RCS file, a script is created to convert some or all of its RCS revisions to ClearCase versions.
- For each specified directory, conversion scripts are created for all the RCS files it contains. Commands are placed in the master conversion script to create a directory element for the specified directory itself, and for its subdirectories (except one named *RCS*).

Each *source-name* must be a simple file or directory name. This enables the conversion scripts to reliably access the source data when they are executed. Specifying the parent directory (..) causes an error, as does any pathname that includes a slash (/) character.

Thus, before entering this command, you should change to the directory where (or under which) the RCS files to be converted reside. If the RCS files reside in *RCS* subdirectories, use the `-r` option to enable *clearcvt_rcs* to find them.

Preservation of RCS Information as Attributes. *Default:* No attributes are attached to versions converted from RCS revisions.

–V Attaches an attribute of type *RCS_REVISION* to each newly-created version. The string value of the attribute is the RCS revision number of the converted revision. (The conversion script creates attribute type *RCS_REVISION*, if necessary.)

Each attribute requires about 1Kb of storage in the VOB database.

–S If an RCS revision's state is not the default (`Exp`), attaches an attribute of type *RCS_REVISION* to the newly-created version. The string value of the attribute is the RCS state attribute of the converted revision.

EXAMPLES

- Create conversion scripts for a single RCS file.

```
% clearcvt_rcs myprogram.c,v
```

- Convert three RCS files in the current working directory to elements in VOB directory */usr/src/project/include*; store the conversion scripts in directory *cvt_include*, and discard the RCS files' history information.

```
% clearcvt_rcs –o cvt_include –n bgr{1,2,3}.h,v
% set S = 'pwd'
% cd /usr/src/project/include
% $S/cvt_include/cvt_script
```

SEE ALSO

cleartool subcommands: chtype, protect, rntype, setview
clearcvt_ccase, clearcvt_dsee, clearcvt_sccs, clearcvt_unix, rcs(1), rsh(1) or remsh(1), sccs(1)

NAME clearcvt_sccs – convert SCCS files to ClearCase elements

SYNOPSIS

clearcvt_sccs [**–I** *date-time* | **–s** *date-time*] [**–r**] [**–n**] [**–o** *script-dir-pname*]
[**–p** *file-pname*] [**–e** *file-pname*] [**–B** *branch-id*] [**–V**]
[**–T** *translation-file*]
[*source-name …*]

DESCRIPTION

Any user can create conversion scripts with this utility. If the '–n' option is not used to create the scripts, only the VOB owner or the 'root' user can run them. We recommend that the VOB owner run the scripts, to minimize the likelihood of permissions problems in multiple-host conversion situations.

clearcvt_sccs converts Source Code Control System (SCCS) files into ClearCase *elements* and *versions*. The source data for a conversion can range from a single file to an entire directory tree.

clearcvt_sccs ignores information in SCCS files that is not related to version-tree structure; this includes flags, id keywords, user lists, and Modification Request numbers. You can use the `–V` option to preserve SCCS-IDs as attributes of the corresponding ClearCase versions.

S-Files, G-Files, and P-Files

clearcvt_sccs works directly with the structured SCCS *s-files*, which have the *s.* file name prefix. It does not convert the *g-files* created with `get` and `get -e` commands; it merely issues warning messages indicating which files are checked out. Be sure to check in such files with the *delta* command before running the converter.

Other than the issuing of warning messages for checked-out files, *clearcvt_sccs* ignores the *p-files* created by `get -e`.

The *s-files* can (but need not) be "buried" in *SCCS* subdirectories; if they are, the subdirectory level is collapsed in the conversion process — for example, SCCS file *./proj/SCCS/s.main.c* becomes element *./proj/main.c*.

Multiple-Pass Conversion

You can convert an SCCS file in several passes. For example, you might use *clearcvt_sccs* to convert major revision level 1, and subsequently use *clearcvt_sccs* again to convert major revision level 2. On the subsequent passes, the conversion scripts will update an existing element correctly if that element has not be modified in the interim.

CONVERSION PROCESS

The conversion process consists of two stages: export and import.

Export Stage

Any user can enter the 'clearcvt_sccs' command that implements the export stage.

The export stage takes place in the area where the SCCS files reside. *clearcvt_sccs* creates a set of *conversion scripts* (Bourne shell scripts), containing commands to create elements, branches, and versions.

In the import stage, you will use the *master conversion script* to invoke all the individual scripts. If any of the files to be converted reside below (rather than *in*) the current working directory, the master conversion script also includes commands to create corresponding directory element(s).

For each SCCS *s-file* it processes, *clearcvt_sccs* extracts versions and writes a conversion script that either

- creates a new element and checks in a version for each SCCS revision, or
- checks out an existing element (optionally, on a branch) and checks in new version for each SCCS revision that has not already been converted

clearcvt_sccs automatically "chases" symbolic links it encounters during the export stage.

Import Stage

Only a VOB's owner or the root user can run the conversion scripts that implement the import stage (unless 'clearcvt_sccs –n' option was used to create the scripts).

The import stage takes place within an existing VOB, in a shell whose view uses the ClearCase default config spec. In this stage, you execute the master conversion script created by *clearcvt_sccs*, populating the VOB with new elements and versions. These changes to the VOB are documented with event records:

- Each time an individual script creates a new file element, an `import file element` event record is stored in the VOB database, along with the standard `create element` event record. It is associated with the parent directory element, not with the new file element itself. Heuristic: the "import" event record is created only if the object is more than 24 hours old.
- Each time a script creates a new version, the standard `create version` event record is annotated with the comment from the SCCS revision.
- The `import file element` event record is always stamped with the current time. The `create version` and `create element` event records are timestamped according to the original SCCS data (unless `clearcvt_sccs -n` was used to create the conversion scripts).
- The event record for the creation of a branch gets the same timestamp as the branch's first version.
- The event record for the attaching of an attribute gets the same timestamp as the associated version.
- Event records for the creation of directory elements and type objects are stamped with the current time.

Incremental Conversion and Restartability. The conversion scripts created by *clearcvt_sccs* can skip certain SCCS revisions, or entire SCCS files, during the import stage. This capability enables valuable features:

- You can convert an SCCS file to a ClearCase element in several conversion passes. You might use incremental conversion for time-budgeting reasons (there are too many revisions to convert all at once), or because the SCCS file is still under development.
- If execution of the master conversion script terminates prematurely for any reason, you can restart it; it will skip revisions it has already converted, effectively resuming where it left off.

CAUTION: Conversion scripts created with `clearcvt_sccs -n` are not restartable.

An import script decides whether to skip an entire SCCS file using a heuristic: if the target branch of the ClearCase element already has a version created at the same time (or later than) the most recent revision in the source SCCS file, the element is skipped. You can suppress this element-skipping algorithm by running the conversion script in an environment where CVT_UPDATE is set to any non-null string. This forces the conversion script to consider converting each SCCS revision.

For each SCCS revision, an import script will decline to create a corresponding version if it already exists on the target ClearCase branch — that is, if it has the same time-modified stamp (or a more recent one). But even when it bypasses version-creation, the import script still updates the version's meta-data (for example, version labels) using information from the SCCS revision.

Background Execution of the Import Stage. You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

SPECIAL CHARACTERS IN FILE NAMES

clearcvt_sccs can handle file names that include some, but not all, of the characters that are special to the UNIX shells. Conversion fails for any file name that includes any of these characters:

```
'   '   "   <Space>   <Tab>   [   ]   ?   *
```

For example:

Succeeds	**Fails**
foo&bar	foo bar
$SCCS_LIB	yellow'sunset
yellow(Y)	dog*bert

VERSION TREE STRUCTURE AFTER CONVERSION

Revisions on the main branch of an SCCS file have two-digit identifiers (for example, `1.2`). These revisions become versions on the *main* branch of the ClearCase element, as illustrated in Figure 15.

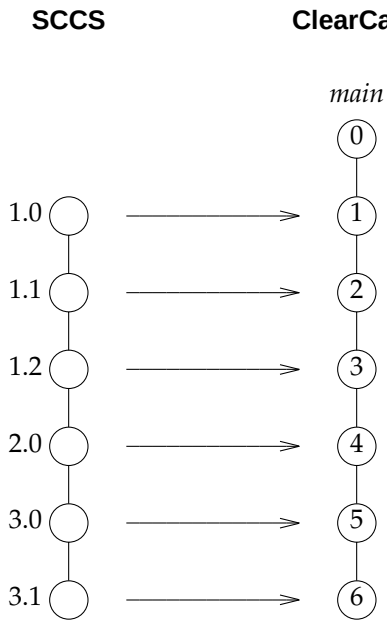


Figure 15. Conversion of SCCS Revisions

Note that the "major revision" substructure in the SCCS revision tree is lost in the translation — all the SCCS revisions become versions on the *main* branch. (But you can use the `-V` option to preserve this information in the form of attributes attached to the versions.)

Revisions on subbranches of an SCCS file have four-digit identifiers (for example, `1.2.1.5`). These revisions become versions on subbranches of the ClearCase element, as illustrated in Figure 16.

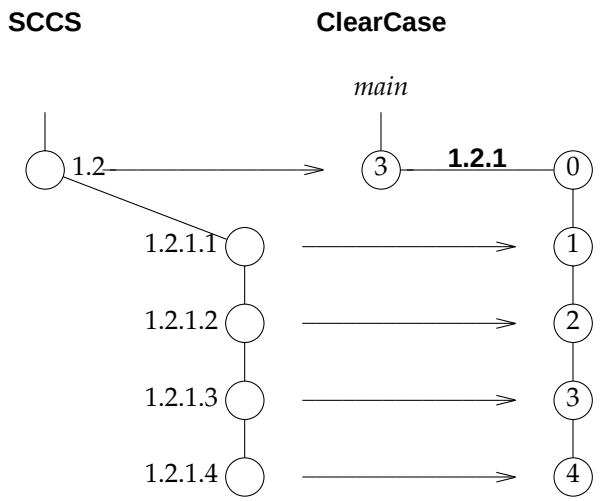


Figure 16. Conversion of SCCS Subbranches

The conversion script creates branch types with three-digit names (`1.2.1` in the example above). Thus, SCCS revision 1.2.1.3 becomes version *3* on branch *1.2.1.*

Branches Off Branches

Although it is not illustrated in the example above, *clearcvt_sccs* can handle SCCS files that include branches off branches.

CONVERTER TRANSLATION FILE

You can use a *converter translation file* to control the names of ClearCase branches created from SCCS branches. If you name such a file, using the **–T** option, *clearcvt_sccs* uses it to:

- Look up each SCCS branch-id to see how to translate it to the name of a branch type. If a match is found, the branch-id is translated the same way.
- Record each translation of a new SCCS branch-id, for use in future lookups.

The first time you use *clearcvt_sccs*, use `–T` to create a new translation file. On subsequent invocations of *clearcvt_sccs*, use `–T` again, specifying the same translation file, for consistent name translation.

Each line of the translation file has three fields, indicating one SCCS-to-ClearCase translation:

1. the keyword `branch` (Translation files for use with some other converters can also have the keyword `version`.)
2. an SCCS branch-id
3. the corresponding ClearCase branch type

Thus, you might create a translation file containing the following line to have all SCCS "3.0.1" branches become ClearCase *phoenix_proj* branches:

```
branch 3.0.1 phoenix_proj
```

OPTIONS AND ARGUMENTS

Storage Location of Conversion Scripts. *Default:* Subdirectory *cvt_dir* is created in the current working directory, and all the conversion scripts are stored there. An error occurs if *./cvt_dir* already exists.

–o *script-dir-pname*
Creates the specified directory and stores the conversion scripts there. An error occurs if the directory already exists.

Handling of Directory Arguments. *Default:* If a UNIX directory is specified as a *source-name* argument: (1) the *s-files* in that directory are converted; (2) a directory element is created for *source-name* and for each of its subdirectories; (3) the contents of the subdirectories are ignored.

–r *clearcvt_sccs* descends recursively into all *source-name* arguments that are directories.

Preprocessing / Postprocessing. *Default:* The import stage consists solely of the execution of the ClearCase-generated commands in the conversion scripts.

–p *file-pname*
Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed before any files are converted.

–e *file-pname*
Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed after all files are converted.

Transcription of History Information. *Default: clearcvt_sccs* extracts historical information from each *s-file* and places it in the conversion script. (It also makes some system calls, such as *getgrgid*(3), to supplement this information.) Thus, the corresponding ClearCase version will have the same *stat*(2) information — user, group, and time–modified stamp — as the original SCCS revision. The `create version` and `create element` event records created by the conversion script also get this original information.

–n Event records and *stat* information for new elements and versions reflect the "who" and "when" of the execution of the conversion script, not the original data.

CAUTION: Using this option creates conversion scripts that are not restartable.

Branch Name Translation. *Default:* As described above in the section "Version Tree Structure after Conversion", *clearcvt_sccs* creates ClearCase branch names based on the SCCS revision IDs.

–T *translation-file*
Uses the specified converter translation file to control the mapping between SCCS branches and ClearCase branches. See the "Converter Translation File" section above.

Selective Conversion of Files. *Default: clearcvt_sccs* converts all SCCS revisions it finds.

–I *date-time* Converts "important" revisions only, but includes *all* revisions created since the specified time. Important revisions are those with labels, those at which branches were created, and those at the ends of branches. The time is specified as follows:

date-time := *date***.***time* | *date* | *time* | **now**

date := *day-of-week* | *long-date*
day-of-week := **today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date := *d*[*d*]–*month*[–[*yy*]*yy*]
month := **January** | ... | **December** | **Jan** | ... | **Dec**
time := *h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

–s *date-time* *This option is designed for regular, incremental updating of an element from an SCCS file that is still under development. Be sure to specify a 'date-time' that covers the entire period since the preceding update. In other situations, it is probably better to use '–I' instead of '–s'.*

Only SCCS revisions created since the specified time are processed. Exception: a branch created at an "old" revision is converted if one or more "new" revisions exist on the branch, or if environment variable CVT_UPDATE is set to a non-null value.

NOTE: In an incremental updating situation, removal of a branch from an SCCS revision is not propagated to the ClearCase element.

Specifying Files to be Converted. *Default:* The current working directory (equivalent to specifying "." as the *source-name* argument). Each *s-file* in the current working directory will be converted; a directory element will be created for each subdirectory of the current working directory (except one named *SCCS*).

source-name …

One or more pathnames, specifying *s-files* and/or directories:

- For each specified *s-file*, a script is created to convert some or all of its SCCS revisions to ClearCase versions.
- For each specified directory, conversion scripts are created for all the *s-files* it contains. Commands are placed in the master conversion script to create a directory element for the specified directory itself, and for its subdirectories (except one named *SCCS*).

Each *source-name* must be a simple file or directory name. This enables the conversion scripts to reliably access the source data when they are executed. Specifying the parent directory (..) causes an error, as does any pathname that includes a slash (/) character.

Thus, before entering this command, you should change to the directory where (or under which) the *s-files* to be converted reside. If the *s-files* reside in *SCCS* subdirectories, use the `–r` option to enable *clearcvt_sccs* to find them.

Selection of SCCS Data to be Converted. *Default:* All SCCS revisions in an *s-file* are converted to ClearCase versions.

–B *branch-id*

Converts only the specified branch of an SCCS file, along with all the ancestor revisions of that branch. You can specify a "true" branch with a three–digit *branch–id*. For example, `–B 2.0.1` would convert all revisions 2.0.1.1, 2.0.1.2, and so on, along with revision 2.0 (where

the branch was created) and all ancestors of revision 2.0 on the main branch of the *s-file*.

Alternatively, you can specify a one-digit major revision number as the *branch-id*. For example, `-B 3` converts the subset of versions on the main branch whose major revision number is 3.

A conversion script created with this option locks the *main* branch of an element after updating it. This prevents potential conflicts on the *main* branch if you subsequently import the entire SCCS file.

Preservation of SCCS-IDs as Attributes. *Default:* No attributes are attached to versions converted from SCCS revisions.

–V Attaches an attribute of type *SCCS_ID* to each newly-created version. The string value of the attribute is the SCCS-ID of the converted SCCS revision. (The conversion script creates attribute type *SCCS_ID*, if necessary.)

Each attribute requires about 1Kb of storage in the VOB database.

EXAMPLES

- Create conversion scripts for a single SCCS file.

```
% clearcvt_sccs s.myprogram.c
```

- Convert three SCCS files in the current working directory to elements in VOB directory */usr/src/project/include*; store the conversion scripts in directory *cvt_include*, and discard the SCCS files' history information.

```
% clearcvt_sccs –o cvt_include –n s.bgr{1,2,3}.h
% set S = 'pwd'
% cd /usr/src/project/include
% $S/cvt_include/cvt_script
```

SEE ALSO

cleartool subcommands: chtype, protect, rntype, setview
clearcvt_ccase, clearcvt_dsee, clearcvt_rcs, clearcvt_unix, events_ccase, rcs(1), rsh(1) or remsh(1), sccs(1)

NAME clearcvt_unix – convert UNIX files to versions of ClearCase elements

SYNOPSIS

clearcvt_unix [**–s** *date-time*] [**–r**] [**–n**] [**–L**] [**–i**] [**–o** *script-dir-pname*]
[**–t** *temp-dir-pname*] [**–p** *file-pname*] [**–e** *file-pname*]
[**–b** *target-branch* [**–v** *version-id*]] [*source-name …*]

DESCRIPTION

Any user can create conversion scripts with this utility. But only the VOB owner or the 'root' user can add elements and versions to a VOB using the conversion scripts. We recommend that the VOB owner run the scripts, to minimize the likelihood of permissions problems in multiple-host conversion situations.

clearcvt_unix can convert standard UNIX files into ClearCase *elements*, and can use files to update existing elements. The source data for a conversion can range from a single file to an entire directory tree.

NOTE: By default, *clearcvt_unix* converts *every* file in the current working directory, including "invisible" files (such as *.exrc*). Be sure to clean up text-editor backup files and other detritus before entering this command.

CONVERSION PROCESS

The conversion process consists of two stages: export and import.

Export Stage

Any user can enter the 'clearcvt_unix' command that implements the export stage.

The export stage takes place in the area where the original UNIX files reside. *clearcvt_unix* creates a set of *conversion scripts* (Bourne shell scripts), containing commands to create elements, versions, and VOB symbolic links. In the import stage, you will use the *master conversion script* to invoke all the individual scripts. If any of the files to be converted reside below (rather than *in*) the current working directory, the master conversion script also includes commands to create corresponding directory element(s).

For each file it processes, *clearcvt_unix* writes a conversion script that either

- creates a new element and checks in its first version, or
- checks out an existing element (optionally, on a branch) and checks in a new version

For each UNIX symbolic link it processes, *clearcvt_unix* places commands in the master conversion script to create a VOB symbolic link. (Alternatively, you can use the –L option to have *clearcvt_unix* "chase links".)

Import Stage

Only a VOB's owner or the root user can run the conversion scripts that implement the import stage (unless 'clearcvt_unix –n' option was used to create the scripts).

The import stage takes place within an existing VOB, in a shell whose view uses the ClearCase default config spec. In this stage, you execute the master conversion script created by *clearcvt_unix*, populating the VOB with new elements, versions, and links. These changes to the VOB are documented with event records:

- Each time an individual script creates a new file element, an `import file element` event record is stored in the VOB database, along with the standard `create element` event record. It is associated with the parent directory element, not with the new file element itself. Heuristic: the "import"" event record is created only if the object is more than 24 hours old.
- Each time a script creates a new version, the standard `create version` event record is annotated with the comment `made from unix file`. (You can specify another comment string with environment variable CVT_REPLACE_COMM.)
- Each time a script creates a new VOB symbolic link, a standard `create symbolic link` event record is created.
- The `import file element` event record is always stamped with the current time. The `create version` and `create element` event records are timestamped according to the original file data (unless `clearcvt_unix -n` was used to create the conversion scripts).
- The event record for the creation of a branch gets the same timestamp as the branch's first version.
- Event records for the creation of directory elements are stamped with the current time.

Restartability and Interruptibility

If the master conversion script was invoked previously but terminated prematurely, it attempts to resume working where it left off. On a subsequent run, a file is converted to a version only if its time-modified stamp is later than all existing versions on the target branch.

CAUTION: Conversion scripts created with `clearcvt_unix -n` create ClearCase versions that are "newer" than all the original files to be converted; thus, such scripts are not restartable.

You can execute the master conversion script in the background by ending the command line with an ampersand character (`&`). But a failure may occur if you attempt to suspend execution by typing a SUSP or SWTCH character (typically, `<Ctrl-Z>`).

SPECIAL CHARACTERS IN FILE NAMES

clearcvt_unix can handle file names that include some, but not all, of the characters that are special to the UNIX shells. Conversion fails for any file name that includes any of these characters:

```
‘   ’   "   <Space>   <Tab>   [   ]   ?   *
```

For example:

Succeeds	**Fails**
foo&bar	foo bar
$UNIX_LIB	yellow'sunset
yellow(Y)	dog*bert

OPTIONS AND ARGUMENTS

Storage Location of Conversion Scripts. *Default:* Subdirectory *cvt_dir* is created in the current working directory, and all the conversion scripts are stored there. An error occurs if *./cvt_dir* already exists.

–o *script-dir-pname*
Creates the specified directory and stores the conversion scripts there. An error occurs if the directory already exists.

Conversion of UNIX Symbolic Links. *Default:* Each UNIX symbolic link is converted to a VOB symbolic link with the same link text.

–L ("chase the link") Converts the object to which a UNIX symbolic link points, instead of converting the link itself.

Creation of Identical Successor Versions. *Default: clearcvt_unix* refuses to create a new version that is identical to its predecessor.

–i Creates a new version, even if it is identical to its predecessor.

Directory for Temporary Files. *Default: clearcvt_unix* and its conversion scripts use */tmp* for temporary files.

–t *temp-dir-pname*
Specifies an alternate temporary file directory. This sets the value of CVT_TEMP_DIRECTORY in the master conversion script, *cvt_script*. As an alternative to using this option, edit *cvt_script* before running it, and change the value of this variable.

Creating New Version on a Branch. *Default:* New versions of a file or directory element are created on the element's *main* branch.

–b *target-branch* [**–v** *version-id*]
Converts each file to a version on branch *target-branch* of the new or existing element. Whenever file conversion creates a new element in the target VOB, the parent directory element is also revised on branch *target-branch*.

If branch type *target-branch* does not already exist in the target VOB, the conversion script creates it. If an existing element already has a branch of this type, the new version will extend this branch; otherwise, the conversion script will create a new branch at version */main/LATEST* (*/main/0* for new elements), unless you also use –v to specify another location.

Handling of Directory Arguments. *Default:* If a UNIX directory is specified as a *source-name* argument: (1) the files in that directory are converted; (2) a directory element is created for *source-name* and for each of its subdirectories; (3) the contents of the subdirectories are ignored.

–r *clearcvt_unix* descends recursively into all *source-name* arguments that are directories.

Preprocessing / Postprocessing. *Default:* The import stage consists solely of the execution of the ClearCase-generated commands in the conversion scripts.

–p *file-pname*
Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed before any files are converted.

–e *file-pname*

Copies the contents of *file-pname* into the master conversion script. This file must contain one or more commands executable in a Bourne shell. The command(s) will be executed after all files are converted.

Transcription of 'stat' Information. *Default:* The conversion scripts create versions with the *stat* information of the original file: user, group, and time-modified stamp. The `create version`, `create element`, and `create symbolic link` event records created by the conversion script also get this original information.

–n Event records and *stat* information for new elements and versions reflect the "who" and "when" of the execution of the conversion script, not the original UNIX link.

The *stat* information for new VOB symbolic links is always taken from the run-time environment, not from the original data.

CAUTION: Using this option creates conversion scripts that are not restartable.

Selective Conversion of Files. *Default: clearcvt_unix* converts all files it encounters.

–s *date-time* Only files modified since the specified moment are processed.

date-time	:=	*date***.***time* | *date* | *time* | **now**
date	:=	*day-of-week* | *long-date*
day-of-week	:=	**today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

Specifying Files to be Converted. *Default:* The current working directory (equivalent to specifying "." as the *source-name* argument). Each file and symbolic link in the current working directory will be converted; a directory element will be created for each subdirectory of the current working directory.

source-name …

One or more pathnames, specifying UNIX files, symbolic links, and/or directories:

- For each specified file, a script is created to convert it to a ClearCase version.
- For each specified directory, conversion scripts are created for all the files and symbolic links it contains. Commands are placed in the master conversion script to create a directory element for the specified directory itself, and for its subdirectories.

Each *source-name* must be a simple file or directory name. This enables the conversion scripts to reliably access the source data when they are executed. Specifying the parent directory (..) causes an error, as does any pathname that includes a slash (/) character.

Thus, before entering this command, you should change to the directory where (or under which) the standard UNIX files to be converted reside. To convert all the files in a single directory, you can change either to that directory, or to its immediate parent.

EXAMPLES

Convert the standard UNIX directory tree */scratch/exper* to a subdirectory of existing VOB directory */src/proj*.

Export Stage:

```
% cd /scratch                                  (go to parent of standard directory tree to be converted)
% clearcvt_unix -r exper                       (create the conversion script)
clearcvt_unix -r gui
VOB directory element ".".
VOB directory element "exper".
Converting element "exper/ar.c" ...
Extracting element history ...
Completed.
Converting element ...
Creating element ...
  :
  :
Converting element ...
Creating element ...
Element "exper/util.c" completed.
Creating script file cvt_dir/cvt_script ...
```

Import Stage:

```
% su                                           (become 'root')
# cleartool setview defvu                      (set a view that has default config spec)
# cd /src/proj                                 (go to VOB directory where data is to be imported)
# /scratch/exper/cvt_dir/cvt_script            (run the master conversion script)
Converting files from /scratch/exper .
You are using the default config_spec
Checked out "./." from version "/main/18".
Created directory element "./exper".
Checked out "./exper" from version "/main/0".
Created element "./exper/ar.c" (type "text_file").
Changed protection on "./exper/ar.c".
  :
  :
Making version one of ./exper/ar.c
  :
  :
Checked out "./gui/browse.c" from version "/main/0".
  :
  :
Checked in "./exper" version "/main/1".
Checked in "./." version "/main/19".
```

SEE ALSO

cleartool subcommands: chtype, protect, rntype, setview
clearcvt_ccase, clearcvt_dsee, clearcvt_rcs, clearcvt_sccs, rcs(1), rsh(1) or remsh(1), sccs(1)

NAME cleardiff – compare or merge text files

SYNOPSIS

- Compare files:

  cleardiff [**–win·dow** | **–tin·y**] [**–dif·f_format** | **–ser·ial_format** | **–col·umns** *n*]
  [**–hea·ders_only** | **–qui·et** | **–sta·tus_only**] [**–b·lank_ignore**] *pname1 pname2 …*

- Merge files:

  cleardiff **–out** *output-pname* [**–bas·e** *pname*] [**–qal·l** | **–abo·rt**]
  [**–win·dow** | **–tin·y**] [**–dif·f_format** | **–ser·ial_format** | **–col·umns** *n*]
  [**–hea·ders_only** | **–qui·et** | **–sta·tus_only**] [**–b·lank_ignore**] *pname1 pname2 …*

DESCRIPTION

cleardiff is a line-oriented file comparison and merge utility with a character-based user interface. It can process up to 32 files.

Alternative interfaces: *cleardiff* can be invoked with the `cleartool diff` subcommand to perform a file comparison, or with the `cleartool merge` subcommand to perform a merge. ClearCase also includes a corresponding GUI tool, *xcleardiff*. This tool can be invoked through *cleartool*, with the *xdiff* and *xmerge* subcommands, and through *xclearcase*.

NOTE: You cannot compare directory versions with *cleardiff*; you must use *diff* or *xdiff*. (These commands first analyze the directory versions, then call on *cleardiff*, using the *type manager* mechanism.)

See the *diff* and *merge* manual pages for discussions of how files are compared and merged.

OPTIONS AND ARGUMENTS

–win·dow
–tin·y (mutually exclusive)

–`window` creates a child process, which displays a side-by-side report in a separate 120-character difference window. The *diff* command returns immediately. To exit the difference window, type a UNIX `INTR` character (typically, `<Ctrl-C>`).

–`tiny` is the same as –`window`, but uses a smaller font in a 165-character difference window.

–dif·f_format
–ser·ial_format
–col·umns *n* (mutually exclusive)

–`diff_format` reports both headers and differences in the same style as UNIX *diff*, and suppresses the file summary from the beginning of the report.

–`serial_format` reports differences with each line containing output from a single file, instead of using a side-by-side format.

–`columns` establishes the overall width of a side-by-side report. The default width is 80 (that is, only the first 40 or so characters of corresponding difference lines appear). If *n* does not exceed the default width, this option is ignored.

–hea·ders_only
–qui·et
–sta·tus_only (mutually exclusive)
NOTE: Any of these options can be invoked with `cleartool diff -options`.

`–headers_only` lists only the header line of each pairwise difference. The difference lines themselves are omitted.

`–quiet` suppresses the file summary from the beginning of the report.

`–status_only` suppresses all output, returning just an exit status: a `0` status indicates that no differences were found; a `1` status indicates that one or more differences were found. This option is useful in shell scripts.

–out *output-pname*
Stores the output of a merge in file *output-pname*. This file is not used for input, and must not already exist.

–bas·e *pname*
Makes file *pname* the base file for the merge. If you omit this option, the *pname1* argument becomes the base file, and a merge automatically runs with the `–qall` option invoked.

–abo·rt
–qal·l (mutually exclusive)
`–abort` is intended for use with scripts or batch jobs that involve merges. It allows completely automatic merges to proceed, but aborts any merge that would require user interaction.

`–qall` turns off automatic acceptance of changes in which only one contributor file differs from the base file. *cleardiff* prompts for confirmation of such changes, just as it does when two or more contributors differ from the base file.

–b·lank_ignore
Causes *cleardiff* to ignore extra white space characters in text lines: leading and trailing white space is ignored altogether; internal runs of white space characters are treated like a single `<Space>` character.

pname1 pname2 …
The pathnames of files to compare or merge. These can be view-extended or version-extended pathnames. Only one such argument is required if you also specify a file with the `–base` option.

EXAMPLES

- Compare the current version of an element with a scratch copy in your home directory.

```
% cleardiff msg.c ~/msg.c.tmp
*******************************
<<< file 1: msg.c
>>> file 2: /net/neptune/vobs/proj/src/msg.c.tmp
*******************************
-------------[changed 5]--------|-----------[changed to 5]------------
    static char msg[256];       |   static char msg[BUFSIZ];
```

```
                                -|-
------------[changed 9-11]------|-----------[changed to 9]------------
             env_user(),        |            env_user(), env_home(), e+
             env_home(),        |-
             env_time() );      |
                                -|
```

- Compare the same files, this time in a separate window and using a small font.
  `% cleardiff –tiny msg.c ~/msg.c.tmp`
- Compare the most recent versions on two branches of an element.
  `% cleardiff util.c@@/main/LATEST util.c@@/main/rel2_bugfix/LATEST`

SEE ALSO

cleartool subcommands: diff, merge, xdiff, xmerge
xcleardiff, type_manager, diff(1)

NAME clearlicense – monitor and control ClearCase license database

SYNOPSIS

clearlicense [**–hos·tid** | **–rel·ease** [*username* | *user-ID*] …]

DESCRIPTION

NOTE: Some ClearCase hosts do not use the Atria-provided licensing scheme.

Reports the status of ClearCase's user licensing facility. You can also use this command to *release* (revoke) one or more users' licenses, making them available to other users.

HOW CLEARCASE LICENSING WORKS

ClearCase implements an "active user" floating license scheme. To use ClearCase, you must obtain a *license*, which grants you the privilege to use ClearCase commands and data on any number of hosts in the local area network. When you run any ClearCase program, it attempts to obtain a license for you. If you get one, you can keep it for an extended period: entering any ClearCase command automatically renews it; but if you don't enter any ClearCase command for a substantial period — by default, 60 minutes — another user can take your license. (Think of this as "use it or lose it".)

One or more hosts in the local area network are designated as ClearCase *license server hosts*. Each of these hosts has a *license database file*, named */usr/adm/atria/license.db*, which contains one or more license entries. Each license entry defines a specified number of licenses, allowing that number of ClearCase users to be *active* at the same time. See *license.db* for a description of the license database file format.

When you first attempt to use ClearCase software on any host in the network, a license-verification check is made:

1. ClearCase software on your host reads the name of a license server host from file */usr/adm/atria/config/license_host*. (The administration directory is */var/adm* on some platforms.)
2. It makes an RPC call to the "license server" process on that license server host, to verify your right to use ClearCase. (The license server process is actually *albd_server*, performing these duties in addition to its other tasks.)
3. The license server process determines your rights, and sends back an appropriate message.
4. Depending on the message sent by the license server, your command either proceeds or is aborted.

Subsequently, similar license-verification checks are performed on a periodic basis. The sections below describe in detail how users get and lose licenses.

License Priorities

Each user can (but need not be) assigned a *license priority* in the license database file. Each user specified in a **–user** line gets a priority number: the first user gets priority 1 (highest priority), the second user gets priority 2, and so on. All users who are not specified in any **–user** line share the lowest priority.

Getting a License

When you first run a ClearCase tool, or first enter a UNIX command to access VOB data though a view, a license-verification request is made. In either of the following cases, you get a license and become an *active* user:

- The current number of active users is less than the maximum number specified by the entry (or entries) in the license database file. In this case, you are simply *granted a license*.
- All licenses are currently in use, but there is a user whose license priority is lower than yours. In this case, you are allowed to *bump* that other user, getting his or her license.

Losing a License

When you get a license, its *timeout* period is set to 60 minutes. (A shorter timeout interval can be configured in the license database file.) As you continue to use ClearCase commands and data, your license is periodically refreshed (the timeout period is set to one hour again). If you do nothing ClearCase-related for an hour, you lose your license — it becomes available to other users.

You can also lose your license before the one-hour timeout:

- As described above, a user with a higher *license priority* can "bump" you.
- You or another user can explicitly *release* (revoke) your license, using `clearlicense -release`.

It is perfectly possible to regain a license immediately after losing it.

License Expiration

Each license entry can have an expiration date. (The expiration time is at 00:00 hours on that date.) After the expiration time, attempts to use a license from that license entry will succeed — with a warning message — during a 24-hour grace period. After that, attempts to use those licenses will fail.

THE CLEARLICENSE REPORT

Following is a typical *clearlicense* report:

```
License server on host "neptune".
Running since Monday 4/04/94 15:53:13.

LICENSES:
     Max-Users  Expires       Password [status]
        19          none      2aae4b60.b4ac4f0f.02 [Valid]

 Maximum active users allowed: 19
 Current active users: 6
 Available licenses: 13

ACTIVE users:
              User   Priority    Time-out in
             smith        2       59 minutes (at 10:44:20)
             jones     none       49 minutes (at 10:34:08)
               akp        3       28 minutes (at 10:13:04)
               adm        1       26 minutes (at 10:10:45)
           jackson     none       23 minutes (at 10:07:27)

License Usage Statistics:
2 licenses revoked today 4/14/94.
0 license requests denied.
0 active users bumped by preferred user.
```

The following sections explain the parts of this report.

License Server Field

The license server is the *albd_server* process on the license server host (`neptune` in the example above). The report lists the time at which this process first processed a license-verification request.

Licenses

The information in this section is gathered from the license entry line(s) in the license database file, */usr/adm/atria/license.db*. Each such `-license` line generates a separate line in this report. The *status* value can be:

`Valid` The expiration date (if any) for this set of licenses has not yet arrived.

`Grace` You are now in the 24-hour grace period following the expiration time.

`Expired` The grace period is over, and this set of licenses has expired.

The *current active users* number summarizes the information in the next section of the report.

Active Users

Each line in this section describes one *active* user. The priority `none` indicates that the user is not specified in any **–user** entry and, thus, has the lowest license priority.

License Usage Statistics

This section lists licensing activity statistics, compiled since the time the license server (*albd_server*) started execution:

- the number of explicit license revocations that have occurred today (with `–release`)
- the number of times a user failed to get a license at all
- the number of times a license was automatically transferred to a higher priority user (*bumping*)

OPTIONS AND ARGUMENTS

Default: A report on licenses and user activity is displayed, in the format described above.

–hos·tid Displays the local host's machine identifier. Use this option when you wish to add licenses to this host's existing license database file, or when you wish to make this host an additional license server host. Enter the output of this command as the "license server host ID" on the License Registration Form to be FAXed to Atria Customer Support.

–rel·ease [*username* | *user-ID*] …
Specifies users (by username or by numeric user-ID) whose licenses are to be revoked. Using `–release` without an argument causes your own license to be revoked. To discourage license battles among users, *albd_server* prevents this option from being used an excessive number of times during any single day.

LICENSING ERRORS

This section describes errors typically encountered in ClearCase licensing.

Problems with License Host File

If the file */usr/adm/atria/config/license_host* does not exist or is empty, this message appears:

```
mvfs: ERROR: view view-tag not licensed!
command-name: .: I/O error
```

In addition, error messages are displayed or are logged to */usr/adm/atria/log/view_log*:

```
Error: You do not have a license to run ClearCase.
```

```
Error: Unable to open file "/usr/adm/atria/config/license_host":
No such file or directory.
```

```
Error: Your license server is not specified.
Create "/usr/adm/atria/config/license_host" and put the license server hostname in it.
```

```
Error: You do not have a license to run ClearCase.
```

Problems with License Server Host

If the license server host specified in the *license_host* file cannot be contacted, this message appears:

```
mvfs: ERROR: view view-tag not licensed!
command-name: .: I/O error
```

In addition, error messages are displayed or are logged to */usr/adm/atria/log/view_log*:

```
Error: Cannot contact license server host "hostname"
defined in file /usr/adm/atria/config/license_host.
```

```
Error: You do not have a license to run ClearCase.
```

Losing a License

If you lose your license while a view is active, this message appears when you try to use ClearCase:

```
mvfs: ERROR: view shtest - all licenses in use!
```

SEE ALSO

albd_server, license.db

NAME clearmake – ClearCase build utility / maintain, update, and regenerate groups of programs

SYNOPSIS

clearmake [**–f** *makefile …*] [**–ukinservdp**]
[**–OTFUVMN**] [**–C** *mode*] [**–J** *num*]
[**–A** *BOS-file*] …
[*macro*=*value …*] [*target_name …*]

DESCRIPTION

clearmake is ClearCase's variant of the UNIX *make*(1) utility. It includes most of the features of UNIX System V *make*(1). It also features *compatibility modes*, which enable you to use *clearmake* with makefiles that were constructed for use with other popular *make* variants.

clearmake features a number of ClearCase-specific extensions:

- **Configuration Lookup** — a build-avoidance scheme that is more sophisticated than the standard scheme based on the time-modified stamps of built objects. It includes automatic *dependency detection*. For example, this guarantees correct build behavior as C-language header files change, even if the header files are not listed as dependencies in the makefile.
- **Derived Object Sharing** — developers working in different views can share the files created by *clearmake* builds
- **Creation of Configuration Records** — "software bill-of-materials" records that fully document a build and support rebuildability

Related Manual Pages

The following manual pages include information related to *clearmake* operations and results:

clearaudit	Alternative to *clearmake* for performing audited builds.
clearmake.options	Describes the use of *build options specification* files (BOS files).
makefile_ccase	Describes *clearmake*-specific makefile facilities.
bldhost	Describes the use of a *build hosts file* to control a distributed build.
bldserver.control	Describes the server-side control facilities for a distributed build.
abe	Describes the *audited build executor* server program that runs on a remote host in a distributed build.
config_record	Describes the *configuration records* created by *clearmake* or *clearaudit*.
derived_object	Describes the *derived objects* created by *clearmake* or *clearaudit*.
lsdo	*cleartool* subcommand to list derived objects created by *clearmake* or *clearaudit*.
catcr, diffcr	*cleartool* subcommands to display and compare configuration records created by *clearmake* or *clearaudit*.

rmdo *cleartool* subcommand to remove a derived object from a VOB.

View Context Required

For a build that uses the data in one or more VOBs, the shell from which you invoke *clearmake* must have a view context — either a *set view* or a *working directory view*. If you have a working directory view, but it differs from your set view, an error occurs.

You can build objects in a standard directory, without a view context, but this disables many of *clearmake*'s special features.

CLEARMAKE AND MAKEFILES

clearmake is designed to read *makefiles* in a way that is compatible with other *make* variants. For details, including discussions of areas in which the compatibility is not absolute, see the *makefile_ccase* manual page.

HOW BUILDS WORK

In many ways, ClearCase builds adhere closely to the standard *make* paradigm:

1. You invoke *clearmake*, optionally specifying the names of one or more *targets*. (Such explicitly-specified targets are termed *goal targets*.)
2. *clearmake* reads zero or more *makefiles* each of which contains targets and their associated *build scripts*. It also reads one or more *build options specification* (BOS) files, which supplement the information in the makefile(s).
3. *clearmake* supplements the makefile-based software build instructions with its own *built-in rules*. (And when it runs in a compatibility mode, *clearmake* also reads a file that defines built-in rules specific to that mode.)
4. For each target, *clearmake* performs *build avoidance*, determining whether it actually needs to execute the associated build script ("perform a target rebuild"). It takes into account both *source dependencies* ("have any changes occurred in source files used in building the target?") and *build dependencies* ("must other targets be updated before this one?").
5. If it decides to perform a target rebuild, *clearmake* executes its build script.

The following sections describe special *clearmake* build features in more detail. Figure 17 illustrates the associated data flow.

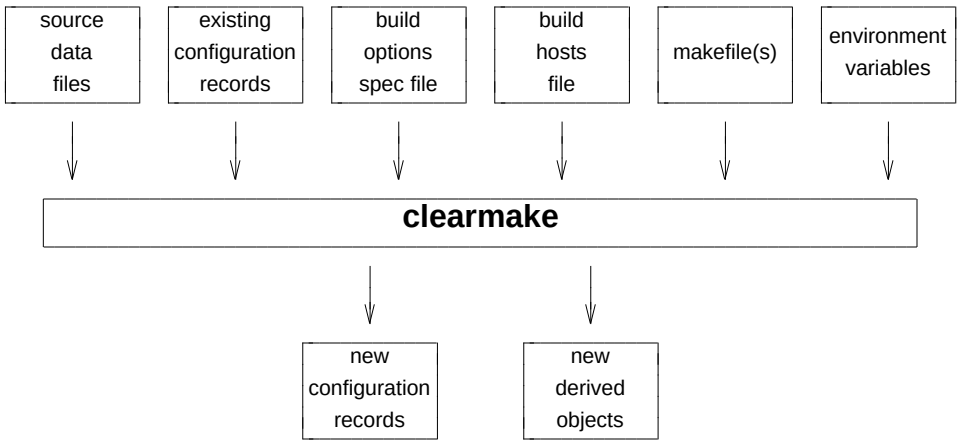


Figure 17. Data Flow in a clearmake Build

CONFIGURATION RECORDS AND DERIVED OBJECTS

In conjunction with the MVFS file system, *clearmake* audits the execution of all build scripts, keeping track of file usage at the OS-system-call level. For each execution of a build script, it creates a *configuration record* (*CR*), which includes the versions of files and directories used in the build, the build script, build options, (for example, macro assignments) and other related information. A copy of the CR is stored in the *VOB database* of each VOB in which the script has built new objects.

A file created within a VOB by a build script is called a *derived object* (*DO*). Each time a derived object is built by some user working in some view, a corresponding VOB database object is also created. This "central accounting" scheme enables any view to access and possibly share — subject to access permissions — any derived object, no matter what view it was originally created in.

For each build script execution, ClearCase logically associates each DO just created with the build script's CR.

You can suppress the creation of CRs and derived objects with the `-F` option. See *config_record* and *derived_object* for details.

(Files created in non-VOB directories are not derived objects — see section "MVFS Files and Non-MVFS Objects" below.)

Configuration Record Hierarchies

A typical makefile has a hierarchical structure. Thus, a single invocation of *clearmake* to build a "high-level" target can cause multiple build scripts to be executed and, accordingly, multiple CRs to be created. See *config_record* for more on *configuration record hierarchies*.

CONFIGURATION LOOKUP AND WINK-IN

For directory targets, *clearmake* uses standard *make* logic.

When a target names a file in a VOB, *clearmake* (by default) uses *configuration lookup* to determine whether a build is required. This involves a comparison of the CRs of existing derived objects with the current *build configuration*:

- the versions of elements selected by the view's config spec
- the build options to be applied, as specified on the *clearmake* command line, in the environment, in makefile(s), or in build options specification file(s) — see "Build Options Specification File" below
- the build script to be executed

In performing configuration lookup, *clearmake* considers a *DO version* (a derived object that has been checked in as a version of an element) only if the version was created "in place". That is, the element's pathname must be the same as the original derived object's.

clearmake first tries to avoid rebuilding by *reusing* a DO in the current view; this succeeds only if the CR of the candidate DO matches the current build configuration. *clearmake* can also avoid rebuilding by finding another DO, built in another view, whose CR matches the current build configuration. In this case, it will *wink-in* that derived object, causing it to be *shared* among views. Other derived objects created by the same build script (termed *siblings*) are winked-in at the same time. *clearmake* rebuilds a target only if it is unable to locate any existing derived object that matches the current build configuration.

Promotion of Data Containers

The first time a derived object is winked-in, *clearmake* invokes the *promote_server* utility to copy the derived object's data (its *data container* file) from view-private storage to a derived object storage pool.

Suppressing Configuration Lookup

You can override the default configuration lookup behavior with command options and ClearCase-specific special targets. For example, `-T` turns off configuration lookup, basing rebuild decisions on time-modified stamps.

MVFS FILES AND NON-MVFS OBJECTS

All files with pathnames below a VOB-tag (VOB mount point) are termed *MVFS files*:

- checked-in versions of file elements (data stored in VOB)
- checked-out versions of file elements (data stored in view)
- other view-private files
- derived objects

Conversely, a *non-MVFS object* is any file, directory, or link whose pathname is not under a VOB-tag; such objects are not version controlled. By default, non-MVFS objects are not audited during *clearmake* builds. A CR will include information on a non-MVFS object used by a build script only if:

- the object appears as an explicit dependency in the makefile, or
- the object can be inferred to be a dependency through *clearmake*'s suffix rules

This kind of dependency is referred to as a *makefile dependency*. For example:

```
src.o : /usr/include/stdio.h
```

Non-MVFS Files in Configuration Lookup

During configuration lookup, *clearmake* examines each non-MVFS file that is listed in the CR of a candidate DO. The CR entry includes: (1) the non-MVFS file's size; (2) its time-modified stamp; (3) its checksum. The "current version" of the non-MVFS file must match the CR entry in one of these ways:

- first check: file size and time-modified stamp
- second check: file size and checksum

OPTIONS AND ARGUMENTS

clearmake supports the options below. In general, "standard" *make* options are lowercase characters; *clearmake* extensions are uppercase. Options that do not take arguments can be "ganged" on the command line (for example, `-rOi`).

–f *makefile* Use *makefile* as the input file. If you omit this option, *clearmake* looks for input files named *makefile* and *Makefile* (in that order) in the current working directory. You can use more than one `-f` *makefile* argument pair. Multiple input files are effectively concatenated.

–u (unconditional) Rebuild all goal targets specified on the command line, along with the recursive closure of their dependencies, regardless of whether or not they need to be rebuilt. (See also `-U`.)

–k Abandon work on the current entry if it fails, but continue on other targets that do not depend on that entry.

–i Ignore error codes returned by commands.

–n (no-execute) List command lines from the makefile for targets which need to be rebuilt, but do not execute them. Even lines beginning with an at-sign (`@`) character are listed. See "Command Echoing and Error Handling" in the *makefile_ccase* manual page.

EXCEPTION: A command containing the string `$(MAKE)` is always executed.

–s (silent) Do not list command lines before executing them.

–e Environment variables override macro assignments within the makefile. (But *macro=value* assignments on the command line or in a *build options spec* override environment variables.)

–r Do not use the built-in rules in file */usr/atria/etc/builtin.mk*.

–v (verbose) Slightly more verbose than the default output mode. Particularly useful features of verbose mode include:

- listing of why *clearmake* does not reuse a DO that already appears in your view (for example, because its CR does not match your build configuration, or because your view does not have a DO at that pathname)
- listing of the names of DOs being created

–d (debug) Quite verbose; appropriate only for debugging makefiles.

–p Lists all target descriptions and all macro definitions, including target-specific macro definitions and implicit rules.

–C *mode* (compatibility) Invokes one of *clearmake*'s compatibility modes. (Alternatively, you can use environment variable CLEARCASE_MAKE_COMPAT to specify a compatibility mode.) *mode* can be:

sgismake Emulates *smake*(1) on IRIX-5 systems. Reads file */usr/include/make/system.mk* instead of */usr/atria/etc/builtin.mk* to define built-in make rules.

sgipmake Emulates *pmake*(1) on IRIX-5 systems. Reads file */usr/include/make/system.mk* instead of */usr/atria/etc/builtin.mk* to define built-in make rules.

sun Emulates the standard *make*(1) on SunOS systems. Reads file */usr/include/make/default.mk* (SunOS-4) or */usr/share/lib/make/make.rules* (SunOS-5) instead of */usr/atria/etc/builtin.mk* to define built-in make rules.

gnu Emulates the Free Software Foundation's *Gnu make* program. The Gnu make built-in rules (which are hard-coded) take effect instead of the rules in file */usr/atria/etc/builtin.mk*.

std Invokes the "standard" *clearmake* — with no compatibility mode enabled. Use this option to nullify a setting of the environment variable CLEARCASE_MAKE_COMPAT.

For details on compatibility mode features, see "Compatibility of clearmake with Other Variants of make" in the *ClearCase User´s Manual*.

–O
–T
–F (mutually exclusive)

–O compares only the names and versions of objects listed in the targets' CRs; it does not compare build scripts or build options. This is useful when this extra level of checking would force a rebuild that you do not want. Examples:

- The only change from the previous build is the setting or cancelling of a "compile-for-debugging" option.
- A target was built using a makefile in the current working directory. Now, you wish to reuse it in a build to be performed in the parent directory, where a different makefile builds the target (with a different script, which typically references the target using a different pathname).

–T makes rebuild decisions using the standard algorithm, based on time-modified stamps; configuration lookup is disabled. (A CR is still created for each build script execution.)

NOTE: This causes both view-private files and derived objects to be used for build avoidance. The CR hierarchy created for a hierarchical build will have a "hole" wherever *clearmake* reuses a view-private file for a subtarget — the view-private file does not have a CR to be included in the CR hierarchy.

–F works like –T, but also suppresses creation of configuration records. All MVFS files created during the build will be view-private files, not derived object.

–U Unconditionally builds *goal targets* only — subtargets undergo build avoidance. If you don't specify any target on the command line, the default target is the goal. (The –u option unconditionally builds both goal targets and build dependencies.)

–V Restricts configuration lookup to the current view only. Wink-in of DOs from other views is disabled.

–M Restricts dependency checking to *makefile dependencies* only — those dependencies declared explicitly in the makefile or inferred from a suffix rule. All *detected dependencies* are ignored. For "safety", this automatically disables wink-in of DOs from other views; it is quite likely that other views select different versions of detected dependencies.

For example, a derived object in your view may be reused even if it was built with a different version of a header file than is currently selected by your view.

–J *num* Enables *clearmake*'s distributed building and parallel building capabilities. The maximum number of concurrent target rebuilds is set to the integer *num*. Special cases:

- If *num*=0, both distributed and parallel building are disabled. (This is equivalent to not specifying a –J option at all.)
- If *num*=1, distributed building is enabled, but only one build script will be dispatched at a time (that is, serial building, not parallel building).

Alternatively, you can specify *num* as the value of environment variable CLEARCASE_BLD_CONC. (See "Parallel and Distributed Building" below.)

–N Disables the default procedure for reading one or more BOS files. See the *clearmake.options* manual page for a discussion of the default procedure.

–A *BOS-file …*

You can use this option one or more times to specify BOS files to be read instead of, or just after, the ones that are read by default. Using –N along with this option specifies "instead of"; omitting –N causes *clearmake* to read the –A file(s) after reading the standard BOS files.

Alternatively, you can specify a colon-separated list of BOS file pathnames as the value of environment variable CLEARCASE_BLD_OPTIONS_SPECS.

MAKE MACROS AND ENVIRONMENT VARIABLES

String-valued variables called *make macros* can be used anywhere in a makefile: in target lists, in dependency lists, and/or in build scripts. For example, the value of make macro CFLAGS can be incorporated into a build script as follows:

```
cc -c $(CFLAGS) msg.c
```

Environment variables (EVs) can also be used in a makefile, but only in a build script. For example:

```
print:
        print_report -style $$PRT_STYLE -dest $${PRT_DEST}.rpt
```

clearmake converts the double-dollar-sign ($$) to a single dollar sign; the EV is expanded in the shell in which the build script executes. (Programs invoked by the build script can also read their environment, using the standard *getenv*(2) system call.)

Conflict Resolution

Conflicts can occur in specifications of make macros and environment variables. For example, the same make macro might be specified both in a makefile and on the command line; or the same name might be specified both as a make macro and as an environment variable.

clearmake resolves such conflicts similarly to other *make* variants:

- Make macros specified on the command line or in a BOS file override any other settings.
- Make macros specified in a makefile have the next highest priority.
- EVs have the lowest priority.

Using the –e option switches the lower two levels — EVs get higher priority than make macros specified in a makefile.

Conflict Resolution Details. The following discussion treats this topic more precisely (but less concisely).

clearmake starts by converting all EVs in its environment to make macros. (SHELL is an exception — see below.) These EVs will also be placed in the environment of the shell process in which a build script executes. Then, it adds in the make macros declared in the makefile. If this produces name conflicts, they are resolved as follows:

- If *clearmake* was not invoked with the –e option, the make macro "wins": the macro value overwrites the EV value in the environment.
- If *clearmake* was invoked with the –e option, the EV "wins": the EV value becomes the value of the make macro.

Finally, *clearmake* adds make macros specified on the command line or in a BOS file; these settings are also added to the environment. These assignments *always* override any others that conflict. (A command-line assignment overrides a BOS setting of the same macro.)

SHELL Environment Variable

clearmake does not use the SHELL environment variable to select the shell program in which to execute build scripts. It uses a Bourne shell (*/bin/sh*), unless you specify another program with a SHELL *macro*: on the command line, in the makefile, or in a build options spec.

MAKEFLAGS Environment Variable

The MAKEFLAGS environment variable provides an alternative (or supplementary) mechanism for specifying *clearmake* command options. It can contain a string of keyletters, the same letters used for command-line options.

For example, these are equivalent:

```
% setenv MAKEFLAGS ei          options set in environment
% clearmake foo
% clearmake -ei foo            options set on command line
```

clearmake combines the value of the environment variable MAKEFLAGS with the options specified on the command line (if any). The combined string of keyletters becomes the value of the macro MAKEFLAGS, available to build scripts.

This is very useful for build scripts that involve recursive invocations of *clearmake*. When `clearmake -n` is applied to such a build script, all the nested invocations of *clearmake* pick up the "no-execute" option from the value of MAKEFLAGS. Thus, no target rebuilds actually take place, even though many levels of *clearmake* command may be executed. This is one way of debugging all of the makefiles for a software project, without actually doing anything.

Option letters that take arguments are not allowed in a MAKEFLAGS string; neither are the `-v` and `-d` options. You can use special EVs, described next, to specify options that are not supported through MAKEFLAGS.

Special Environment Variables

The environment variables described below are also read by *clearmake* at startup. In some cases, as noted, you can also specify the information as a make macro — on the command line, in a makefile, or in a BOS file.

CLEARCASE_BLD_AUDIT_TMPDIR
: Sets the directory where *clearmake* creates temporary build audit files. If this variable is not set, *clearmake* creates these files in */tmp*. All temporary files are deleted when *clearmake* exits. CLEARCASE_BLD_AUDIT_TMPDIR must not name a directory under a VOB–tag; if it does, *clearmake* prints an error message and exits.

CLEARCASE_BLD_CONC
: Sets the concurrency level. This EV takes the same values as the `-J` option. Specifying a `-J` option on the command line overrides the setting of this EV.

CLEARCASE_BLD_HOST_TYPE
: Determines the name of the *build hosts file* to be used during a distributed build (`-J` option): file *.bldhost.$CLEARCASE_BLD_HOST_TYPE* in your home directory. (Your home directory is determined by examining the password database.)

  C SHELL USERS: Set this EV in your *.cshrc* file, not in your *.login* file. The distributed build facility invokes a remote shell, which does not read the *.login* file.

  CLEARCASE_BLD_HOST_TYPE can also be coded as a make macro.

CLEARCASE_BLD_OPTIONS_SPECS
: A colon-separated list of pathnames, each of which specifies a BOS file to be read. You can use this EV instead of specifying BOS files on the command line with one or more `-A` options.

CLEARCASE_BLD_SHELL_FLAGS
: Specifies command options to be passed to the subshell program that executes a build script command. Default: `-e`.

CLEARCASE_BLD_UMASK
Sets the *umask*(1) value to be used for newly-created derived objects. It may be advisable to have this EV be more permissive than your standard umask — for example, CLEARCASE_BLD_UMASK = 2 where umask = 22. The reason to create DOs that are more accessible than other files is *wink-in*: a winked-in file retains its original ownership and permissions. For example, when another user winks-in a file that you originally built, the file is still owned by you, is still a member of your principal group, and still has the permissions with which you created it. (NOTE: You can use the standard *chmod* command to change the permissions of a DO after you create it.)

CLEARCASE_BLD_UMASK can also be coded as a make macro.

CLEARCASE_BLD_VERBOSITY
Sets the *clearmake* message logging level, as follows:

1	equivalent to using `-v` on the command line
2	equivalent to using `-d` on the command line
0 or undefined	equivalent to standard message logging level

If you also specify `-v` or `-d` on the command line, the higher value prevails.

CLEARCASE_INCLUDE_CXX_RULES
Expands to an `include` statement that reads in file *$(CLEARCASE_MAKE_CONFIG_DIR)/cxx.mk*. This file contains definitions that are used in a C++ environment to handle parameterized types (templates).

CLEARCASE_MAKE_COMPAT
Specifies one of *clearmake*'s compatibility modes. This EV takes the same values as the `-C` option. Specifying a `-C` option on the command line overrides the setting of this EV.

CLEARCASE_MAKE_CONFIG_DIR
Expands to the full pathname of the *clearmake* configuration directory in the ClearCase installation area — typically */usr/atria/config/clearmake*.

ADDITIONAL CLEARMAKE EXTENSIONS

The following sections describe additional ClearCase-specific features supported by *clearmake*.

Build Options Specification File

A *build options specification* (BOS) file is a text file containing macro definitions and/or ClearCase-specific special targets. We recommend that you place nonpermanent option specifications (for example, a macro that specifies "compile for debugging") in a BOS file, instead of on the *clearmake* command line. This minimizes the likelihood of having *clearmake* perform a rebuild "unexpectedly" (for example, because you specified `-g` on the command line last time, but forgot to specify it this time).

See *clearmake.options* for details.

Parallel and Distributed Building

clearmake supports *parallel building* (execution of several build scripts concurrently) and *distributed building* (use of one or more remote hosts to execute build script). Most often, these features are used in combination — for example, performing a hierarchical build by running build scripts on many hosts at the same time.

These features are enabled by the `-J` option, which specifies the parallelism (concurrency) level, and the *build hosts file*, which lists hosts where build scripts can be dispatched.

Use these features as follows:

1. Make sure that CLEARCASE_BLD_HOST_TYPE is set, as described in section "Special Environment Variables" above.
2. Create the appropriate *build hosts file* in your home directory, containing the names of hosts to be used in a distributed build. For example, you might create file *.bldhost.irix5* with a list of hosts to be used when building the IRIX-5 variant of a target. The administrator of a build host machine can control and limit its accessibility. See *bldhost* and *bldserver.control* for details.
3. Use the `-J` option to set the maximum number of target rebuilds to be executed concurrently.

Before starting a parallel build, *clearmake* determines what work needs to be done, organizing it as a sequence of target rebuilds. It then dispatches build scripts to hosts, using a *load balancing* scheme. By default, a host will be used only if it is at least 50% "idle". You can adjust this idleness threshold with a `-idle` specification in your build hosts file. See *bldhost* and *bldserver.control* for details.

Parallel building can be suppressed for all of a makefile's targets with the special `.NOTPARALLEL` target. See *makefile_ccase* for details.

Remote Build Environment. *clearmake* dispatches a build script to a remote host by invoking a remote shell there. This shell, in turn, runs an *audited build executor* (*abe*) process which executes the build script. *abe* runs in a minimal environment — the remote shell command does not export the current environment to the remote host. You can use make macros on the command line, but not environment variables, to control remote execution of build scripts. Example:

```
env A=xxx clearmake foo B=yyy C=zzz
```

When executed on a remote host, the build script for target *foo* will always see the settings of *B* and *C*; but it will not see the setting of *A* if this variable is also set in the shell's startup environment on the remote host. For example, you cannot set the build script's search path on the remote host like this:

```
env PATH=... clearmake foo
```

The preferred way to pass such settings is in a BOS file. (See the *abe* and *clearmake.options* manual page for more.)

Terminal Output. In a serial build (`-J` not specified or `-J 1`), a target's build script is connected to *stdout* directly. Output appears as soon as it is produced by the script's commands. In a parallel build (`-J` specified with an argument ≥ 2), the standard output of each build script is accumulated in a temporary file by *clearmake*. When the build script finishes, *clearmake* sends it to *stdout* all at once.

Disabling of Parallel Distributed Building. Any of the following conditions disables parallel distributed building, causing target rebuilds to be performed serially on the local host only:

- You do not specify the `-J` option on the *clearmake* command line.
- Environment variable CLEARCASE_BLD_HOST_TYPE is not set when you invoke *clearmake*.

- Your *build hosts file* (*.bldhost.$CLEARCASE_BLD_HOST_TYPE* in your home directory) cannot not be read.

A particular host will not be used during a distributed build if your current view cannot be used on that host. (Perhaps the host cannot access the view storage directory.) Likewise, a host will not be used if its *abe* cannot be successfully started.

Build Reference Time

clearmake takes into account the fact that software builds are not instantaneous. As your build progresses, other developers can continue to work on their files, and may check in new versions of elements that your build uses. If your build takes an hour to complete, you would not want build scripts executed early in the build to use version 6 of a header file, and scripts executed later to use version 7 or 8. To prevent such inconsistencies, *clearmake* "locks out" any version that meets both these conditions:

- The version was checked in after the moment that the build began — the *build reference time*.
- The version is now selected by a config spec rule that involves the *LATEST* version label.

This reference-time facility applies to checked-in versions of elements only; it does not lock out changes to checked-out versions, other view-private files, and non-MVFS objects. *clearmake* automatically adjusts for the fact that the system clocks on different hosts in a network may be somewhat out of sync (*clock skew*).

The "Build Sessions" section below includes additional information regarding build reference time.

Build Sessions

A "top-level" invocation of *clearmake* or *clearaudit* starts a *build session*. The time at which the build session begins becomes the build reference time for the entire build session.

A build session can have any number of *subsessions*, all of which inherit the reference time of the build session. A subsession corresponds to a "nested build" or "recursive make", started when a *clearmake* or *clearaudit* process is invoked in the process family of a higher–level *clearmake* or *clearaudit*. Examples of *clearmake* invocations that start subsessions:

- entering a *clearmake* command in a shell started with *clearaudit*
- including a *clearmake* command in a makefile build script executed by *clearmake*

A typical subsession begins while a higher-level session is in the process of performing target rebuilds, each of which has its own build audit. The subsession conducts its *own* build audit(s), independently of the audit(s) of the higher-level session — the audits are not nested or related in any way, other than that they share the same build reference time.

Following are some important ramifications of this architecture:

Element Versions Created During a Build Session. Any version created during a build session and selected by a *LATEST* config spec rule will not be visible in that build session. For example, a build might checkin a derived object it has created; subsequent commands in the same build session will not "see" the checked-in version, unless it is selected by a config spec rule that does not involve the version label *LATEST*.

Coordinating Reference Times of Several Builds. Different build sessions have different reference times. The "best" way to have a series of builds share the same reference time is to structure them as subtargets of a single build target in a makefile. An alternative approach is to run all the builds within the same *clearaudit* session. For example, you might write a shell script, *multi_make*, that includes several invocations of *clearmake* (along with other commands). Running the script as follows ensures that all the *clearmake* builds will be subsessions that share the same reference time:

```
clearaudit -c multi_make
```

Objects Written at More than One Level. Undesirable results occur when the same file is written at two or more session levels (for example, a top-level build session and a subsession): the build audit for the higher-level session does not contain complete information about the file system operations that affected the file. Example:

```
clearaudit -c "clearmake shuffle > logfile"
```

The file *logfile* may be written both:

- during the *clearaudit* build session, by the shell program invoked from *clearaudit*
- during the *clearmake* subsession, when the *clearaudit* build session is suspended

In this case, *clearaudit* issues this error message:

```
Unable to create derived object "logfile"
```

To work around this limitation, "postprocess" the derived object at the higher level with a copy command:

```
clearaudit -c "clearmake shuffle > log.tmp"
cp log.tmp logfile
rm log.tmp
```

No Automatic Creation of Configuration Record Hierarchy. CRs created during a build session and its subsessions are not automatically linked into a single configuration record hierarchy. The *config_record* manual page presents techniques for accomplishing such linkage.

EXIT STATUS

clearmake returns a zero exit status if all goal targets are successfully processed. It returns a nonzero exit status in two cases:

- *clearmake* itself detects an error, such as a syntax error in the makefile. In this case, the error message includes the string "clearmake".
- A makefile build script terminates with a nonzero exit status (for example, a compiler error). In this case, the error message includes the name of the program that encountered the error, such as "cc".

EXAMPLES

- Unconditionally build the default target in a particular makefile, along with all its dependent targets.

  ```
  % clearmake –u –f project.mk
  ```

- Build target *hello* without checking build scripts or build options during configuration lookup. Be moderately verbose in generating status messages.

  ```
  % clearmake –v –O hello
  ```

- Build the default target in the default makefile, with a particular value of make macro INCL_DIR. Base rebuild decisions on time-modified comparisons instead of performing configuration lookup, but still produce CRs.

```
% clearmake –T INCL_DIR=/usr/src/include_test
```

- Perform a parallel, distributed build of target *bgrs*, using up to five of the hosts listed in file *.bldhost.solaris* in your home directory.

```
% setenv CLEARCASE_BLD_HOST_TYPE solaris
% clearmake –J 5 bgrs
```

- Build target *bgrs*, restricting configuration lookup to the current view only. Have environment variables override makefile macro assignments.

```
% clearmake –e –V bgrs
```

- Build the default target in Sun compatibility mode.

```
% clearmake –C sun
```

FILES

/usr/atria/etc/builtin.mk

SEE ALSO

"Building Software with ClearCase" chapter in the *ClearCase Concepts Manual*
ClearCase User´s Manual
abe, bldhost, bldserver.control, clearaudit, cleartool, clearmake.options, config_spec, derived_object, makefile_ccase promote_server, scrubber
umask(1)

NAME clearmake.options – clearmake build options specification file (BOS)

SYNOPSIS

One or more files read by 'clearmake', specifying make macros and special targets

DESCRIPTION

A *build options specification* (BOS) file is an ASCII file containing macro definitions and/or ClearCase-specific special targets. We recommend that you place "temporary" macros (such as `CFLAGS=-g` and others not to be included in a makefile permanently) in a BOS file, rather than specifying them on the *clearmake* command line.

By default, *clearmake* reads one or more BOS files in this order:

- the file *.clearmake.options* in your home directory (as indicated in the password database) — the place for macros to be used every time you execute *clearmake*
- "local" BOS file(s), each of which corresponds to one of the makefiles specified with a `-f` option, or read automatically by *clearmake*. Each BOS file has a name in this form:

  makefile-name.**options**

  Examples:

  ```
  makefile.options
  Makefile.options
  project.mk.options
  ```

Multiple local BOS files are read in the same order as the `-f` options. If a macro is defined in multiple BOS files read by *clearmake*, the last definition wins. No error occurs if a particular BOS file does not exist. You can override this default behavior with the *clearmake* command-line options `-N` and/or `-A`, or with the environment variables MAKEFLAGS and CLEARCASE_BLD_OPTIONS_SPECS. *clearmake* displays the names of the BOS files it reads if you specify the `-v` or `-d` option, or if $CLEARCASE_BLD_VERBOSITY ≥ 1.

The following sections describe the various kinds of BOS file entries.

Standard Macro Definitions

A standard macro definition has the same form as a make macro defined in a makefile:

macro_name **=** *string*

For example:

```
CDEBUGFLAGS = -g
```

Target-Dependent Macro Definitions

A target-dependent macro definition takes this form:

target-list **:=** *macro_name* **=** *string*

Any standard macro definition can follow the `:=` operator; the definition takes effect only when targets in *target-list* and their dependencies are processed. Targets in the *target-list* must be separated by white space. For example:

```
foo.o bar.o := CDEBUGFLAGS=-g
```

Two or more higher-level targets can have a common dependency. If the targets have different target-dependent macro definitions, the dependency is built using the macros for the *first* higher–level target *clearmake* considered building (whether or not it actually built it).

Shell Command Macro Definitions

A shell command macro definition replaces a macro name with the output of a shell command:

macro_name **:sh =** *string*

This defines the value of *macro_name* to be the output of *string*, an arbitrary shell command. In command output, all `<NL>` characters are replaced by `<Space>` characters. For example:

```
BUILD_DATE :sh = date
```

Special Targets

These ClearCase-specific special targets can be used in a build options spec:

```
.NO_CONFIG_REC   .NO_CMP_SCRIPT   .NO_WINK_IN   .NO_CMP_NON_MF_DEPS
```

See the "Special Targets" section of the *makefile_ccase* manual page for descriptions of these targets.

Include Directives

To include one BOS file in another, use the `include` or `sinclude` ("silent include") directive. For example:

```
include /usr/local/lib/aux.options
sinclude $(OPTS_DIR)/clearmake.options
```

Comments

A BOS file can contain comment lines, which begin with a pound sign (#) character.

MAKE MACROS AND ENVIRONMENT VARIABLES

Make macros set in a BOS file override make macro definitions in a makefile, and also override environment variables. But macros specified on the *clearmake* command line override those set in a BOS file. All BOS file macros (except those overridden on the command line) are placed in the build script's environment. If a build script recursively invokes *clearmake*:

- The higher-level BOS file setting (now transformed into an EV) will be overridden by a make macro set in the lower-level makefile; but if the recursive invocation used the *clearmake*'s `–e` option, the BOS file setting prevails.
- If another BOS file (associated with another makefile) is read at the lower level, its make macros override those from the higher-level BOS file.

SEE ALSO

clearmake, abe, makefile_ccase, bldhost, bldserver.control

NAME clearprompt – prompt for user input

SYNOPSIS

- Prompt for text:

  clearprompt text −pro·mpt *prompt_string* **−out·file** *pname* [**−mul·ti_line**]
  [**−def·ault** *string* | **−dfi·le** *pname*] [**−pre·fer_gui**]

- Prompt for pathname:

  clearprompt file −pro·mpt *prompt_string* **−out·file** *pname* [**−def·ault** *filename* | **−dfi·le** *pname*]
  [**−pat·tern** *match_pattern*] [**−dir·ectory** *dir_path*] [**−pre·fer_gui**]

- Prompt for continue-processing choice:

  clearprompt proceed −pro·mpt *prompt_string* [**−typ·e** *type*] [**−def·ault** *choice*]
  [**−mas·k** *choice*[**,***choice*]] [**−pre·fer_gui**]

- Prompt for yes-no choice:

  clearprompt yes_no −pro·mpt *prompt_string* [**−typ·e** *type*] [**−def·ault** *choice*]
  [**−mas·k** *choice*[**,***choice*]] [**−pre·fer_gui**]

  proceed *choice* is one of: **proceed**, **abort**

  yes_no *choice* is one of: **yes**, **no**, **abort**

  type is one of: **ok**, **warning**, **error**

DESCRIPTION

Prompts the user for input, then either stores the input in a file or returns an appropriate exit status. *clearprompt* is designed for use in trigger action scripts. (See the *mktrtype* manual page.) Another use for *clearprompt* is in scripts that are called from *xclearcase group files* (see also "Customizing the Graphical Interface" in the *ClearCase User´s Manual*).

clearprompt can interact with the user either through *stdin* and *stdout* (*CLI mode*), or through a pop-up window (*GUI mode*). It uses the latter style automatically when a trigger fires on an operation invoked through the GUI program *xclearcase*.

A trigger action script (or any other script) can use the exit status of `clearprompt proceed` or `clearprompt yes_no` to perform conditional processing:

User selects	**Exit status**
yes	0
proceed	0
no	1
abort	2

If an error occurs in *clearprompt* itself, the exit status is an integer greater than 9.

OPTIONS AND ARGUMENTS

text [**−mul·ti_line**]
file
proceed
yes_no (mutually exclusive) Specifies the kind of user input to be prompted for:

`text` prompts for a single text line (with *no* trailing `<NL>` character). `text -multi_line` works just like *cleartool* comment input: the user can enter any number of lines, ending with `.<Return>` or `<Ctrl-D>`.

`file` prompts for a file name or, if `−prefer_gui` is specified, pops up a File Browser window.

`proceed` prompts for a choice between the alternatives `proceed` and `abort`.

`yes_no` prompts for a choice among the alternatives `yes`, `no`, and `abort`.

−pro·mpt *prompt_string*
Specifies a message to be displayed, presumably explaining the nature of the interaction.

−out·file *pname*
Specifies the file to which the user's input will be written.

−def·ault *string*
Specifies the text to be written to the `−outfile` file if the user simply presses `<Return>` (in CLI mode) or clicks the "Ok" button (in GUI mode).

−dfi·le *pname*
A variant of `−default`: reads the default text from a file instead of the command line.

−def·ault *choice*
Specifies the choice made if the user simply presses `<Return>` (in CLI mode) or clicks the "Ok" button (in GUI mode). Be sure to include *choice* in the `−mask` list, as well.

−typ·e *type* Specifies the *severity level*: `ok`, `warning`, or `error`. The only effect is in the way the user is prompted for input.

−mas·k *choice*[*,choice*]]
Restricts the universe of choices for a `proceed` or `yes_no` interaction. For example, the following command restricts a `yes_no` interaction to the choices `yes` and `abort` (`no` is excluded):

```
clearprompt yes_no -mask yes,abort ...
```

−pat·tern *match_pattern*
−dir·ectory *dir_path*
When `clearprompt file` executes in GUI mode, the File Browser window contains a pathname filter. By default, this window displays the names of all files in the current working directory. You can use the `−directory` and/or `−pattern` option to specify a different directory and/or a file name pattern (for example, `*.c`) to restrict which file names are displayed. You can change the filter after the File Browser appears.

–pre·fer_gui

Causes *clearprompt* to try to work in GUI mode; but if the attempt to pop up an interaction window fails, falls back to CLI mode.

EXCEPTION: If *clearprompt* is invoked by a trigger firing on an *xclearcase* (not *cleartool*) operation, GUI mode is *forced*. If an interaction window cannot be created, an error occurs.

EXAMPLES

NOTE: See the *mktrtype* manual page for additional examples.

- Prompt the user to enter a name, writing the user's input to file *uname*. Use the value of the USER environment variable if the user simply presses `<Return>`.

```
% clearprompt text –outfile uname –default $USER –prompt "Enter User Name:"
```

- Ask a question and prompt for a yes/no response, using a separate window if possible. Make the default response "no".

```
% clearprompt yes_no –prompt "Do You Want to Continue?" \
      –default no –mask yes,no –prefer_gui
```

- Prompt for a file name, using a separate window if possible. Restrict the choices to files with a `.c` suffix, and write the user's selection to a file named *myfile*.

```
% clearprompt file –prompt "Select File From List" \
      –outfile myfile –pattern '*.c' –prefer_gui
```

SEE ALSO

cleartool subcommands: mktrigger, mktrtype
xclearcase

NAME config_ccase – ClearCase configuration files

SYNOPSIS

files in /usr/adm/atria (or a subdirectory therein), used by ClearCase server processes to configure system operation

DESCRIPTION

ClearCase processes create and consult the files described in the sections below.

Files in /usr/adm/atria

Anyone can the information in this directory, but only the *root* user can modify it.

.albd_well_known_port

An empty flag file created by a host's *albd_server* process when it is first started. This file is created if, and only if, the host's services database (file */etc/services* or NIS map *services)* lists the `albd` service at the standard port number, 371.

If the flag file exists on a host, a ClearCase client program running on the host uses this standard port number to contact *albd_server* programs throughout the network; otherwise, the client must look up the port number of the `albd` service in the services database. Note that this scheme requires that the `albd` service be registered at the same port number throughout the network.

almd_times
almd_startup_time

These files are used by ClearCase Release 2.0 and later to support usage of VOBs created with earlier releases.

license.db (license server host only) The *license database file*, which defines a set of ClearCase licenses. See the *license.db* manual page.

no_mvfs_tag

A flag file created during ClearCase installation; the ClearCase startup/shutdown script uses this file to determine whether to perform operations involving the MVFS (multiversion file system).

Files in /usr/adm/atria/config

Anyone can read and write information in this directory, in order to configure the local host.

config/alternate_hostnames

If a host has two or more network interfaces (two or more separate lines in the */etc/hosts* file or the hosts NIS map), you must create a file with this name on that host to record its multiple entries. For example, suppose that the */etc/hosts* file includes these entries:

```
159.0.10.16   widget       sun-005 wid
159.0.16.103  widget-gtwy  sun-105
```

In this case, the *alternate_hostnames* file should contain:

```
widget
widget-gte
```

Note that only the first hostname in each hosts entry need be included in the file. The file must list each alternative hostname of a separate line. There is no commenting facility; all lines are significant. If a host does not have multiple network interfaces, this file should not exist at all.

config/automount_prefix
By default, *automount*(1M) mounts directories under */tmp_mnt*. If another location is used for a host's automatic mounts (for example, you use `automount -M`, or you use an alternative auto-mount program), then you must specify it in file */usr/adm/atria/automount_prefix*. For example, if your automatic mounts take place within directory */autom*, create an *automount_prefix* file containing this line:

```
/autom
```

config/bldserver.control
Controls the way in which a host is used during distributed builds. See the *bldserver.control* manual page.

config/license_host
(required for each ClearCase host) Contains the name of the host that acts as the ClearCase *license server host* for the local host

Files in /usr/adm/atria/cache

Information written and used by local server processes.

cache/Clearcase_check
A subdirectory, populated with zero-length files, used for ClearCase licensing.

cache/clearcase_specdev
A symbolic link that points to the device on which the MVFS performs *ioctl*(2) system calls. This file is created by *mount_mvfs* when the *viewroot* directory (by default, */view*) is mounted; it is deleted when the viewroot is unmounted. If this link is missing or points to the wrong place, commands will display this error message:

```
cleartool: Error: Unable to open file "viewroot": ClearCase object not found.
```

cache/scrubber_fs_info A cache of file system usage statistics maintained by the *scrubber* utility. This file is used to implement *scrubber*'s free-space-analysis heuristic.

SEE ALSO

albd_server, bldserver.control, license.db, mount_mvfs

NAME config_record – bill-of-materials for clearmake build or clearaudit shell

DESCRIPTION

A *configuration record* (CR) is a meta-data item that contains information gathered in a ClearCase *build audit*. The *clearmake* build utility performs a build audit — in conjunction with the *multiversion file system* (MVFS) on a ClearCase client host — during execution of a target rebuild. Typically, this involves execution of a single build script; for a "double-colon" target, the rebuild may involve execution of multiple build scripts.

clearaudit enables build auditing during the execution of an arbitrary program — typically a shell.

One CR is written to a VOB database each time a target rebuild creates one or more *derived objects* within that VOB. (If a build script creates derived objects in multiple VOBs, a copy of the CR is written to each VOB database.) A configuration record is logically associated, and can be accessed through, all the derived objects created during the build audit.

MVFS OBJECTS AND NON-MVFS OBJECTS

In a configuration record, two kinds of file system objects are distinguished.

- An *MVFS object* is a file or directory "in a VOB" — one whose pathname is below a VOB-tag (VOB mount point).
- A *non-MVFS object* is an object not accessed through a VOB (compiler, system-supplied header file, temporary file, and so on).

CONTENTS OF A CONFIGURATION RECORD

A configuration record both provides a build's "bill of materials" and documents its "assembly procedure". A CR can include several sections; if created by *clearaudit*, it does not include sections related to build scripts.

Header Section

The Header section of a CR includes five lines; we use an example to describe these lines:

`Target util.o built by akp.dvt`
: The makefile target associated with the build script (`ClearAudit_Shell` for a CR produced by *clearaudit*); the user who started the build.

`Host 'neptune' running IRIX 5.x (IP12)`
: The host on which the build script was executed, along with some *uname*(2) information.

`Reference Time 15-Sep-93.08:18:56, this audit started 15-Sep-93.08:19:00`
: A timestamp indicating the build's *reference time*: the time at which *clearmake* or *clearaudit* began execution. In a hierarchical build, involving execution of multiple build scripts, all the resulting CRs share the same reference time. (For more on reference time, see the *clearmake* manual page.)

  This line also includes a timestamp indicating when the build script for this particular CR began execution.

`View was neptune:/usr/people/akp/views/930825.vws`
: The view in which the build took place, identified by the location of the view storage directory.

`Initial working directory was /usr/hw/src`
: The current working directory at the time build script execution (or *clearaudit* execution) began.

MVFS Objects Section

The MVFS Objects section of a CR includes:

- Each MVFS file or directory read during the build — both versions of elements and view-private files used as build input. It includes checked-out versions of file elements.
- Each derived object produced by the target rebuild.

Non-MVFS Objects Section

The Non-MVFS Objects section of a CR includes each non-MVFS file that appears as an explicit dependency in the makefile, identified by DTM stamp.

If there are no such files, this section is omitted. It is always omitted from a CR produced by *clearaudit*.

Variables and Options Section

The Variables and Options section of a CR lists the values of make macros defined during the build. If no make macros were defined, this section is omitted.

This section is omitted from a CR produced by *clearaudit*.

Build Script Section

The Build Script section of a CR lists the script that was read from a makefile and executed by *clearmake*.

This section is omitted from a CR produced by *clearaudit*.

CONFIGURATION RECORD HIERARCHIES

A typical makefile has a hierarchical structure. Thus, a single invocation of *clearmake* to build a "high-level" target can cause multiple build scripts to be executed and, accordingly, multiple CRs to be created. Such a set of CRs can form a *configuration record hierarchy*, which reflects the structure of the makefile.

For example, consider the makefile in Figure 18.

```
hello: hello.o msg.o libhello.a                    ——— top-level target
        cc -o hello hello.o util.o msg.o
        date > /tmp/flag.hello

hello.o:                                           ┐
        cc -c hello.c                              │
                                                   │
msg.o:                                             │ second-level targets
        cc -c msg.c                                │
                                                   │
libhello.a: user.o env.o                           │
        ar r libhello.a user.o env.o               ┘

user.o:                                            ┐
        cc -c user.c                               │ third-level targets
                                                   │
env.o:                                             │
        cc -c env.c                                ┘
```

Figure 18. Hierarchical Makefile

A complete build of target `hello` produces the CR hierarchy shown in Figure 19.

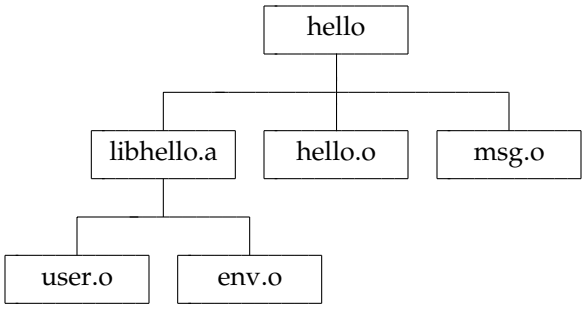


Figure 19. CR Hierarchy Created by Complete Build: 'clearmake hello'

An individual "parent-child" link in a hierarchy is established in either of these ways:

- **In a target/dependencies line** — For example, the following target/dependencies line declares derived objects *hello.o*, *msg.o*, and *libhello.a* to be build dependencies of derived object *hello*:

  ```
  hello: hello.o msg.o libhello.a
          ...
  ```

  Accordingly, the CR for *hello* is the parent of the CRs for the *.o* files and the *.a* file.

- **In a build script** — For example, in the following build script, derived object *libhello.a* in another directory is referenced in the build script for derived object *hello*:

  ```
  hello: $(OBJS)
          cd ../lib ; $(MAKE) libhello.a
          cc -o hello $(OBJS) ../lib/libhello.a
  ```

Accordingly, the CR for *hello* is the parent of the CR for *libhello.a*.

NOTE: The recursive invocation of *clearmake* in the first line of this build script produces a separate CR hierarchy, which is not necessarily linked to the CR for *hello*. It is the *second* line of the build script that links the CR for *../lib/libhello.a* with that of *hello*.

PHYSICAL STORAGE OF CONFIGURATION RECORDS

Logically, a configuration record is stored in some VOB database (and, sometimes, with additional copies in other VOB databases). But the actual contents of a CR migrates from view storage to VOB storage, along with the data container(s) of its associated derived object(s). (See "Physical Storage of Data Containers / Promotion" in the *derived_object* manual page.)

When a derived object is first created in some view, both its data container and its associated configuration record are stored in the view's private storage area. The CR is stored in the view database, in compressed format. To speed configuration lookup during subsequent builds in this view, a compressed copy of the CR is also cached in a view-private file, *.cmake.state*, located in the directory that was current when the build started.

When the DO is *winked-in* to another view, or is checked in as a *DO version*:

- *promote_server* is invoked to copy the data container to a VOB storage pool.
- The configuration record is moved from the view's private storage area to the VOB database.

The process of winking-in an entire set of *sibling* DOs may involve making copies of the CR in multiple VOB databases.

SEE ALSO

cleartool subcommands: catcr, diffcr, lsdo
clearmake, clearaudit, derived_object, makefile_ccase, view_scrubber

NAME config_spec – rules for selecting versions of elements to appear in a view

SYNOPSIS

- Standard Rule:

  scope pattern version-selector [*optional-clause*]

- Time Rule:

  time *date-time*

- File-Inclusion Rule:

  include *config-spec-pname*

DESCRIPTION

A view's *config spec* (short for "configuration specification") contains an ordered set of rules for selecting *versions* of ClearCase *elements*. The view's associated *view_server* process "populates" a view with versions by dynamically evaluating the config spec rules. Each time a reference is made to a file or directory element — either by ClearCase software or by standard programs — the *view_server* uses the config spec to select a particular version of the element. (In practice, a variety of caching techniques and optimizations reduce the computational requirements.)

Config Spec Storage / Default Config Spec

Each view is created with a copy of the system-wide default config spec, */usr/atria/default_config_spec*:

```
element * CHECKEDOUT          For any element, select the checked out version, if any, ...
element * /main/LATEST        ... or else, select most recent version on the 'main' branch
```

Modifying this file changes the config spec that newly-created views will get, but does not affect any existing view.

An individual view's config spec is stored in its *view storage directory*, in two forms:

- **'Source' format** — The "user-visible" version, *config_spec*, contains just the series of config spec rules.
- **'Compiled' format** — A modified version, *.compiled_spec*, which includes ClearCase-internal accounting information. This version is created and used by the *view_server* process.

Do not modify either of these files directly; instead, use the commands listed below. Different views' config specs are independent: they might contain the same set of rules, but changing one view's config spec never affects any other view.

Commands for Maintaining Config Specs

ClearCase commands for manipulating config specs include:

catcs List a view's config spec.

setcs Make a specified file a view's config spec.

edcs Revise the current config spec of a view.

HOW A CONFIG SPEC MAKES OBJECTS VISIBLE

For each element, the following procedure determines which version, if any, appears in the view.

1. The view's associated *view_server* process tries to find a version of the element that matches the first rule in the config spec:
   - If such a version exists, that version appears in the view.
   - If multiple versions match the rule, an error occurs, and no version of the element appears in the view. ClearCase commands that access the element will get errors like this:

     ```
     cleartool: Error: Trouble looking up element "util.c" in directory ".".
     ```

     Standard commands that access the element will get errors like this:

     ```
     UX:cat: ERROR: Cannot open util.c: I/O error
     ```

   - If no version matches the first rule, the search continues.
2. If no matching version was found for the first rule, the *view_server* tries to find a version that matches the second rule.
3. The *view_server* continues in this way until it finds a match, or until it reaches the last rule.

Order is Important

Because the rules in a config spec are processed in order, varying the order may affect version selection. For example, suppose this rule appears near the beginning of a config spec:

```
element * /main/LATEST
```

Any subsequent rules in the config spec will never be used, because the rule will always provide a match — every element has a most-recent version on its *main* branch.

Failure to Select Any Version

If no version of an element matches any rule in the config spec, then:

- The element's data will not be accessible through the view. The standard *ls* command and other standard programs will get a `not found` error when attempting to access the element.
- The ClearCase *ls* command will list the element with a `[no version selected]` annotation. You *can* specify the element in commands that access the VOB database only, such as *describe*, *lsvtree*, and *mklabel*.

View-Private Files

A view's config spec has no effect on the *private* objects in a view, such as view-private files, links, directories, or derived objects. View-private objects are always visible.

EXCEPTION: If a config spec lacks a "CHECKEDOUT" rule, the view-private file that is a file element's checked-out version will not be visible. See section "Special Version Selectors" below.

OVERALL SYNTAX GUIDELINES

Each config spec rule must be contained within a single physical text line; you cannot use a backslash (\) character to continue a rule onto the next line. Multiple rules can be placed on a single line, separated by semicolon (;) characters.

Lines that begin with a pound sign (#) character are comments.

Extra white space (<Space>, <Tab>, vertical-tab, and form-feed) characters are ignored, except within the version selector. If a version selector includes white space, enclose it in single-quote (') characters.

STANDARD RULES

A standard version-selection rule takes this form:

scope pattern version-selector [*optional-clause*]

The following subsections describe these components.

Scope

The *scope* specifies that the rule applies to all elements, or restricts the rule to a particular type of element.

element The rule applies to all elements.

element –file
The rule applies to *file* elements only. This includes any element created with a *mkelem* command that omits `-eltype directory` (or a user-defined element type derived from *directory*).

element –directory
The rule applies to *directory* elements only. This includes any element created wit *mkdir* or `mkelem -eltype directory` (or a user-defined element type derived from *directory*).

element –eltype *element-type*
The rule applies only to elements of the specified element type (predefined or user-defined). This mechanism is not hierarchical: if element type *aaa* is a supertype of element type *bbb*, the scope `element -eltype aaa` does not include elements whose type is *bbb*. To specify multiple element types, you must use multiple rules:

```
element -type aaa RLS_1.2
element -type bbb RLS_1.2
```

Selecting Versions of VOB Symbolic Links. There is no "VOB symbolic link" scope. A VOB symbolic link is cataloged (listed) in one or more versions of a directory element. The link appears in a view if:

- One of those directory versions is selected by the view's config spec, and …
- The config spec includes *any* "element" rule — even a `–none` rule.

Pattern

A pathname pattern, which can include any ClearCase *wildcard* — see the *wildcards_ccase* manual page for a complete list. Examples:

`*` Matches all element pathnames.

`*.c` Matches all element pathnames with a `.c` suffix.

`src/util.c`
Matches any element named *util.c* that resides in any directory named *src*.

`/vobs/project/include/util.h`
Matches one particular element.

`src/.../util.c`
Matches any element named *util.c* that resides anywhere within the subtree of a directory named *src* (including in *src* itself).

`src/.../*.[ch]`
Matches all elements with `.c` and `.h` suffixes located in or below any directory named *src*.

`src/...` Matches the entire directory tree (file elements and directory elements) under any directory named *src*.

NOTE: In non-config-spec contexts, the `...` pattern matches directory names only.

Restrictions:

- A view-extended pathname pattern is not valid.
- A relative pathname pattern must start *below* the VOB-tag (VOB mount point, VOB root directory). For example, if the VOB-tag is */vobs/project*, then *project/include/utility.h* is not a valid pattern.
- A full pathname pattern must specify a location beneath a valid VOB-tag. The *setcs* or *edcs* command fails if it encounters an invalid location in any config spec rule:

```
cleartool: Error: Cannot locate vob mount directory: "..."
```

Version Selector

You can use a *version label*, *version–ID*, or any other standard ClearCase *version selector*. See the *version_selector* manual page for a complete list. Examples:

`/main/4` Version 4 on an element's *main* branch.

`REL2` The version to which version label *REL2* has been attached. An error occurs if more than one version of an element has this label.

`.../mybranch/LATEST`
The most recent version on a branch named *mybranch*; this branch can occur anywhere in the element's version tree.

`/main/REL2`
The version on the *main* branch to which version label *REL2* has been attached.

`{QA_Level>3}`
The version to which attribute *QA_Level* has been attached with a value greater than 3. An error occurs if more than one version satisfies this query.

`.../mybranch/{QA_Level>3}`
The most recent version on a branch named *mybranch* satisfying the attribute query.

Standard version selectors cannot select checked-out versions in a config spec rule. (They can in other ClearCase contexts, such as the *find* command.) Instead, you must use the special version selector, *CHECKEDOUT*, described below.

Special Version Selectors. The following special version selectors are valid only in a config spec rule, not in any other ClearCase version-selection context:

CHECKEDOUT
Matches the checked-out version of an element, if this view has a pending checkout. It doesn't matter where (on which branch of the element) the checkout occurred; there is no possibility of ambiguity, since only one version of an element can be checked out to a particular view.

This special version selector actually matches the "checkout-out version" object in the VOB database, which is created by the *checkout* command.

For file elements, standard commands access the view-private file created by *checkout* at the same pathname as the element.

–config *do-pname* [**–select** *do-leaf-pattern*] [**–ci**]
This special version selector replicates the configuration of versions used in a particular *clearmake* build. It selects versions listed in one or more *configuration records* associated with a particular derived object: the same set of versions as would be listed by a `catcr -flat` command. See the *catcr* manual page for explanations of the specifications that follow the `–config` keyword.

During execution of a *setcs* or *edcs* command, the *view_server* resolves the *do-pname* with respect to the view's pre-existing config spec, *not* based on any preceding rules in the config spec being evaluated.

If the configuration record(s) list several versions of the same element, the most recent version is selected to appear in the view. A warning message is displayed in such cases (at the time the config spec is set).

–none Causes an ENOENT (`No such file or directory`) error to occur when a standard UNIX program references the element. In particular:

- No error occurs when a standard *ls* command lists the element's entire parent directory; the element is included in such a listing. This also applies to other *readdir*() situations, such as expansion of wildcard characters and *emacs* file name completion.
- An error does occur when a standard *ls* command names the element explicitly (perhaps after wildcard expansion), or whenever the name is processed with *stat*(2): in an `ls -F` command, when the entire directory is listed with `ls -l`, and so on.
- The ClearCase *ls* command always lists the element, annotating it with `no version selected`.
- In ClearCase commands, the element's standard pathname refers to the element itself. (`–none` suppresses ClearCase's *transparency* mechanism — translation of an element's standard pathname into a reference to a particular version.)

–error Like –none, except that the annotation generated by the ClearCase *ls* command is `error on reference`.

Optional Clause

Some config spec rules can include an additional clause, which modifies the rule's meaning.

–time *date-time*

Modifies the meaning of the special version label *LATEST*: the rule selects from a branch the last version that was created before a particular time. The *date-time* argument is specified in the standard ClearCase format:

date-time	:=	*date***.***time* | *date* | *time* | **now**
date	:=	*day-of-week* | *long-date*
day-of-week	:=	**today** | **yesterday** | **Sunday** | .. | **Saturday** | **Sun** | .. | **Sat**
long-date	:=	*d*[*d*]–*month*[–[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]]

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, it defaults to `00:00:00`. If you omit the *date*, it defaults to `today`. If you omit the century, year, or a specific date, the most recent one is used. Dates before January 1, 1970 UCT are invalid.

The creation times of the versions on the branch are looked up in their `create version` event records. (No error occurs if you use a `–time` clause in a rule that does not involve the version label *LATEST*; the clause has no effect.)

The `–time` clause in a particular rule overrides any general *time rule* currently in effect. (See section "Time Rules" below.)

Restriction: `–time` must precede any other optional clauses.

Examples:

`/main/LATEST –time 10–Jul.19:00`
Most recent version on main branch, as of 7 PM on July 10.

`.../bugfix/LATEST –time yesterday`
Most recent version on a branch named *bugfix* (which can be at any branching level), as of the beginning of yesterday (12 AM).

`/main/bugfix/LATEST –time Wed.12:00`
Most recent version on subbranch *bugfix* of the *main* branch, as of noon on the most recent Wednesday.

`–time 5–Dec.13:00`
December 5, at 1 PM

`–time 11:23:00`
Today, at 11:23 AM

`–time 12–jun–92`
June 12, 1992, at 00:00 AM

`–time now` Today, at this moment.

The *date/time* specification is evaluated on each execution of *setcs* or *edcs*. Thus, the meaning of a relative specification, such as `today`, changes from execution to execution.

–nocheckout
Disables checkouts of elements selected by the rule.

–mkbranch *branch-type-name*
Implements ClearCase's *auto-make-branch* facility. When a version selected by this rule is checked out:

1. A branch of type *branch-type-name* is created at that version.
2. Version 0 on the new branch is checked out, instead of the originally-selected version.

(This is a slight oversimplification. See section "Multiple-Level Auto-Make-Branch" below.) A *mkelem* command invokes the auto-make-branch facility if the config spec includes a "/main/LATEST" rule with a `–mkbranch` clause.

Restrictions: You cannot use `–mkbranch` in combination with `–none` or `–error`.

Multiple-Level Auto-Make-Branch

A config spec can include a "cascade" of auto-make-branch rules, causing *checkout* to create multiple branching levels at once. *checkout* keeps performing auto-make-branch until version 0 on the newly-created branch is *not* selected by a rule with a `–mkbranch` clause; then, it checks out that version. For example:

```
element * CHECKEDOUT                                    (1)
element * .../br2/LATEST                                (2)
element * .../br1/LATEST -mkbranch br2                  (3)
element * MYLABEL -mkbranch br1                         (4)
element * /main/LATEST                                  (5)
```

If you checkout an element in a view that currently selects the version labeled *MYLABEL*:

1. A branch of type *br1* is created at the *MYLABEL* version (Rule 4).
2. Rule 3 now selects the newly-created version *.../br1/0*, so a branch of type *br2* is created at that version.
3. Version *.../br1/br2/0* is checked out. The checked-out version has the same contents as the *MYLABEL* version, and is selected by Rule 1. When you edit and *checkin* a new version, *.../br1/br2/1*, the view will select it with Rule 2.

TIME RULES

A *time rule* takes this form:

time *date-time*

It is analogous to the optional `-time` clause described above. A time rule modifies the meaning of the special version label *LATEST* in subsequent rules, except that:

- An optional `-time` clause in a particular rule overrides any general time rule currently in effect.
- A subsequent time rule cancels and replaces an earlier one.

FILE-INCLUSION RULES

A file-inclusion rule takes this form:

include *config-spec-pname*

The argument specifies a text file containing one or more config spec rules (possibly other `include` rules). Include files are re-read on each execution of *setcs* or *edcs*.

EXAMPLES

- Include a standard set of rules to be used by every user on a particular project:

```
include /proj/cspecs/v1_bugfix_rules
```

- Modify the meaning of "most recent" to mean "as of 7PM on July 10".

```
time 10-Jul.19:00
element /vobs/atria/lib/* .../new/LATEST
element * /main/LATEST
```

- Select version 3 on the *main* branch of a particular header file.

```
element /usr/project/include/utility.h /main/3
```

- Select the most recent version on the *main* branch for all elements with a `.c` file name suffix

```
element *.c /main/LATEST
```

- Select the most recent version on the *bugfix* branch.

```
element * .../bugfix/LATEST
```

- Select versions of elements from a particular development branch, or with a related label.

```
element * CHECKEDOUT
                                  If no checked–out version, select latest version on the 'maint' branch, which may or may not be
element * .../maint/LATEST        a direct subbranch of 'main'
element * BL2.6                   Else, select version labeled 'BL2.6' from any branch
element * /main/LATEST
```

- Select versions of C language source files (`.c` file extension) based on the value of an attribute. A config spec such as this might be used by a developer to select versions of files for which he is responsible.

```
element * CHECKEDOUT
                                        For any '.c' file, select latest version on main branch for which 'jpb' is responsible
element -file *.c /main/{RESPONSIBLE=="jpb"}
                                        Else, select version labeled BL2.6 on main branch from /project/utils directory, or any of its
                                        subdirectories
element -file /project/utils/.../*.c /main/BL2.6
element * /main/LATEST
```

- Use the –mkbranch qualifier to create a new *BL3* branch automatically. Create the branch off the version labeled *BL2.6*, or the latest version on the *main* branch if no version is labeled *BL2.6*.

```
element * CHECKEDOUT
element * .../bl3_bugs/LATEST                  If no version is checked out, select latest version on 'bl3_bugs' branch
                                                Else, select version labeled 'BL2.6' and create new 'bl3_bugs' branch on
element -file * BL2.6 -mkbranch bl3_bugs        checkout
                                                Else, select latest version on 'main' branch and create new branch on checkout
element -file * /main/LATEST -mkbranch bl3_bugs
element * /main/LATEST
```

- Select the version labeled *REL3* for all elements, preventing any checkouts to this view:

```
element * REL3 -nocheckout
```

- Select the most recent version on the *bug_fix_v1.1.1* branch, making sure that this is a subbranch of *bug_fix_v1.1*, which is itself a subbranch of *bug_fix_v1*.

```
element * CHECKEDOUT
element * .../bug_fix_v1.1.1/LATEST
element * .../bug_fix_v1.1/LATEST -mkbranch bug_fix_v1.1.1
element * .../bug_fix_v1/LATEST -mkbranch bug_fix_v1.1
element * /main/LATEST -mkbranch bug_fix_v1
```

When a user checks out an element for which none of these branches yet exist, a "cascade" of *auto-make-branch* activity takes place:

```
% cleartool checkout –nc .
Created branch "bug_fix_v1" from "." version "/main/0".
Created branch "bug_fix_v1.1" from "." version "/main/bug_fix_v1/0".
Created branch "bug_fix_v1.1.1" from "." version "/main/bug_fix_v1/bug_fix_v1.1/0".
Checked out "." from version "/main/bug_fix_v1/bug_fix_v1.1/bug_fix_v1.1.1/0".
```

FILES

/usr/atria/default_config_spec
view-storage-directory/config_spec
view-storage-directory/.compiled_spec

SEE ALSO

cleartool subcommands: catcr, catcs, checkout, checkin, edcs, ls, mkbranch, mkbrtype, mkelem, mkeltype, setcs
clearmake, query_language, version_selector, view_server, wildcards_ccase
csh(1)

NAME crontab_ccase – ClearCase crontab scripts

SYNOPSIS

scripts installed during ClearCase installation and executed periodically by cron(1M)

DESCRIPTION

ClearCase uses the UNIX *cron*(1) utility to perform periodic maintenance of various system files and data structures on each host. *crontab*(1M) scripts are provided for the maintenance of:

- the local host's ClearCase error log files
- the cleartext and derived object storage pools of each VOB that resides on the local host
- the collection of event records stored in the database of each VOB that resides on the local host

ClearCase installation creates or updates directory */usr/atria/config/cron*, with the files described in the following sections.

NOTE: If you modify *ccase_cron.day* or *ccase_cron.wk*, it will be moved aside to a file name with the `.0` (old) extension the next time you install ClearCase.

Crontab Entries for 'root' User

ClearCase installation creates file *crontab.root*, and also appends its contents to the *crontab*(1) file of the *root* user. The crontab entries invoke daily and weekly ClearCase maintenance scripts.

NOTE: If ClearCase is not installed at */usr/atria* on your host, revise *root*'s crontab file after installation to correct the pathnames.

Daily Processing: ccase_cron.day

The script *ccase_cron.day* is run daily from the *root* user's crontab. It, in turn, invokes these other scripts:

scrubber_day.sh — Runs the *scrubber* utility on all VOBs that reside on the local host. This reclaims disk space from VOB cleartext and derived object storage pools, by deleting unneeded data container files. (The definition of "unneeded" is pool-specific; see the *scrubber* and *mkpool* manual pages.)

ccase_local.day — Use this file to customize daily *crontab* processing. If the file exists, it is executed as a Bourne shell script. It is neither created nor modified during ClearCase installation.

Weekly Processing: ccase_cron.wk

The script *ccase_cron.wk* is run weekly from the *root* user's crontab. It, in turn, invokes these other scripts:

cleanlogs.sh — Moves each existing log file to *logfile_name**.old***, and creates a new, empty file, named *logfile_name*, in its place. (See the *errorlogs_ccase* manual page.)

vob_scrubber.sh — Invokes the *vob_scrubber* utility to remove unwanted event records from VOBs that reside on the local host. See the *vob_scrubber* manual page for descriptions of the configuration files that control event-record scrubbing.

ccase_local.wk — Use this file to customize weekly *crontab* processing. If the file exists, it is executed as a Bourne shell script. It is neither created nor modified during ClearCase installation.

LOCAL CRONTAB SCRIPTS AND INSTALLATION OPTIONS

ClearCase may be installed on your host in such a way that directory */usr/atria/config/cron* is not local:

- It may be accessed through a symbolic link to the network-wide *release host*.
- It may be in a remote file system that is mounted on your host.

In such cases, the *ccase_local.day* and *ccase_local.wk* scripts may be shared among several hosts. You may wish to use these "shared local" scripts to invoke "truly local" scripts. For example, you might place this code in *ccase_local.day*:

```
if [ -d /usr/adm/atria/cron ] ; then
    for SCRIPT in /usr/adm/atria/cron/*.day  ; do
            /bin/sh $SCRIPT
    done
fi
```

FILES

/usr/atria/config/cron/crontab.root
/usr/atria/config/cron/ccase_cron.day
/usr/atria/config/cron/scrubber_day.sh
/usr/atria/config/cron/ccase_local.day
/usr/atria/config/cron/ccase_cron.wk
/usr/atria/config/cron/cleanlogs.sh
/usr/atria/config/cron/vob_scrubber.sh
/usr/atria/config/cron/ccase_local.wk
/etc/rc

SEE ALSO

cleartool subcommands: mkpool
cron(1), crontab(1M), errorlogs_ccase, init(1M), rc(8), scrubber, vob_scrubber, view_server

NAME db_dumper, db_loader – dump/load a VOB database schema

SYNOPSIS

invoked as needed by cleartool's 'reformatvob' subcommand

DESCRIPTION

These programs are called by the *reformatvob* command to update the *schema* of a *VOB database*:

- The *db_dumper* program converts binary VOB database files to ASCII files.
- The *db_loader* program reads the ASCII files, creating a new VOB database that uses the up-to-date schema.

reformatvob invokes a VOB's own copy of *db_dumper*: when the VOB is created with *mkvob*, a *db_server* running on the VOB host copies file */usr/atria/etc/db_dumper* into the new VOB's database subdirectory, and changes its access mode to 4555. The *db_server* runs as root; thus, the VOB's copy of *db_dumper* becomes a *setUID-root* program.

By contrast, loading does *not* involve a VOB-specific program. No matter what VOB is being processed, *reformatvob* invokes the same program: */usr/atria/etc/db_loader*. This is also a *setUID-root* program. (Running *site_prep* on the network-wide release host sets the permissions on the original; installation on an individual host preserves the permissions; see the *ClearCase Notebook* for details.)

'db_dumper' PERMISSIONS PROBLEMS

Entering a *reformatvob* command may fail with a message that the copy of *db_dumper* stored within the VOB storage directory has the wrong permissions and/or ownership:

```
cleartool: Error: Database dumper "[VOB-STORAGE-DIR]/db.reformat/db_dumper"
must be setUID and owned by the super-user.
```

Note that the pathname to *db_dumper* is a location within the VOB's database subdirectory, which has been renamed by *reformatvob* to *db.reformat*. Enter the following commands to fix the problem; be sure to enter the pathname of the *db_dumper* program exactly as it appears in the error message.

```
% su
Password: <enter root password>
#chown root [VOB–STORAGE–DIR]/db.reformat/db_dumper
#chmod 4555 [VOB–STORAGE–DIR]/db.reformat/db_dumper
#exit
```

'db_loader' PERMISSIONS PROBLEMS

The *db_loader* program will not be *setUID-root*, and thus will not work correctly, if:

- file */usr/atria/etc/db_loader* is actually located on a remote host, and
- the local host accesses this program through a file system mount that uses a `nosuid` option.

SEE ALSO

cleartool subcommands: reformatvob
ClearCase Notebook

NAME db_server – ClearCase database server program

SYNOPSIS

invoked as needed by the 'albd_server' program

DESCRIPTION

A host's *db_server* processes handle *VOB database* transactions on that host, in response to requests from ClearCase *client* processes throughout the network: *cleartool*, *xclearcase*, *clearmake*, or *abe*. These program do not access VOB databases directly. Instead, they send database transaction requests to a *db_server* process, which runs on the host where the VOB storage area resides (the *VOB host*). The *db_server* process, running as the *root* user, performs the actual database access. Database transactions include:

- creating and modifying *meta-data* (such as attaching a label to a version)
- reading meta-data (such as finding the labels attached to a version)
- writing event records (such as the one that records a *checkout* command)
- writing configuration records
- reading event records and configuration records

Each *db_server* process services a single client (at a time), but can operate on any number of VOBs. A client establishes a connection to a *db_server* with the help of the *albd_server* on the VOB host. The connection is made either with an available *db_server*, or with a newly created one. The connection is broken when the client exits (or fails to make a database transaction over an extended period). At that point, the *db_server* becomes available for use by another client; eventually, an unconnected *db_server* is terminated by *albd_server*.

ERROR LOG

The *db_server* sends warning and error messages to */usr/adm/atria/log/db_server_log*.

SEE ALSO

abe, albd_server, init_ccase, scrubber, view_server, clearmake, cleartool, nfsd(1M)

NAME derived_object – file built by clearmake or clearaudit, with an associated configuration record

DESCRIPTION

In their everyday work, developers think of a derived object as a file created within a VOB directory during *clearmake*'s execution of a build script (or a file created within a VOB directory during execution of an *audited shell* invoked with *clearaudit*).

This manual page uses terminology more rigorously, distinguishing a derived object from its data container:

- A *derived object* (DO) is a VOB database object created by *clearmake* or *clearaudit* to record the creation of a new file within a VOB directory.
- A derived object has an associated data file, its *data container*, which holds the development data written to the file during *clearmake*/*clearaudit* execution. The data container is an ordinary file, not a VOB database object.

DERIVED OBJECT CONTENTS

A derived object in a VOB database contains this information:

file name and VOB directory
: The simple file name under which the data file was created, along with an internal identifier for the directory element in which the file was created. If the derived object was built in a view-private subdirectory — for example, *sun5/util.o* — its location is recorded by a relative pathname, not a simple file name.

  If hard links were made to the derived object during the build, the additional filename/VOB-directory pairs are also recorded in the VOB database.

DO-ID
: A unique identifier, which distinguishes this DO from others created (in other views and/or at other times) at the same pathname. A *DO–ID* includes the extended naming suffix (default: `@@`), a timestamp, and a numeric suffix that guarantees uniqueness. Examples:

```
@@14-Sep.09:54.418
@@13-Sep.09:30.404
@@02-Sep.16:23.353
```

storage location of data container
: – for an unshared DO, a pointer to the view where the data container is stored
  – for a shared DO, a pointer to its derived object storage pool, along with a relative pathname within that pool

pointer to configuration record
: Shared DO only: an internal pointer to the derived object's associated *configuration record* (CR), a meta-data item stored in the same VOB database. (The CR of an unshared derived object is stored in the view where it was built.)

reference count
: The number of times the derived object appears in ClearCase views throughout the network (along with identifiers for those views). A DO's reference count is incremented when it is created, and whenever the DO is *winked-in* to another view; it is also incremented when a

view-private hard link is made to the DO. (For example, the command `ln hello hw` increments the reference count of derived object *hello*.)

Shared and Unshared DOs

When a derived object is first created by a process running in some view:

- It appears only in that view (reference count = 1)
- Its data container is a file in the view's private storage area.

Such a derived object is termed an *unshared* DO.

The first time a derived object is winked-in to another view, its status changes from unshared to *shared*:

- It now appears in two views (and has a reference count of 2).
- Its data container is *promoted* to a VOB storage pool. (See "Physical Storage of Data Containers / Promotion" below.)

Thereafter, the derived object remains *shared*, no matter how many times it is winked-in to additional views, and even if subsequent rebuilds or deletion commands cause it to appear in only one view.

Physical Storage of Data Containers / Promotion

When an unshared derived object is winked-in to another view, *clearmake* invokes *promote_server* to copy (not move) the DO's data container from view storage to VOB storage. The view to which the DO is winked-in uses the data container in VOB storage, as will all other views to which the DO is subsequently winked-in. But the original view continues to use the data container in view storage. (The *view_scrubber* utility removes this "asymmetry", causing *all* views to use the data container in VOB storage.)

Uniqueness of DO-IDs

A derived object's identifier (DO-ID) is unique in that it is guaranteed to differ from the DO-ID of all other derived objects. But it can change over time:

- When a DO passes its first birthday, the timestamp in its DO-ID changes, to indicate the year it was created:

```
when first created:       util.o@@15-Jul.8896
after a year:             util.o@@15-Jul-1992.8896
```

- When a VOB's database is processed with *reformatvob*, all DO-IDs get new numeric suffixes:

```
before 'reformatvob':     util.o@@15-Jul.8896
after 'reformatvob':      util.o@@15-Jul.734
```

Thus, preserving DO-IDs in files or scripts may be of limited use. A derived object gets a truly permanent identifier when it is checked in as a version of an element. See "DO Versions" below.

MANIPULATING DERIVED OBJECTS WITH STANDARD COMMANDS

Modifying a derived object in any way using a standard command or program converts it from a DO to a view-private file:

```
% cleartool ls msg.o                    ('msg.o' is a derived object)
msg.o@@10-Mar.15:33.333
```

```
% touch msg.o                    (use standard command to modify the file)
                                 ('msg.o' is no longer a derived object in this view; reference count decre-
% cleartool ls msg.o             mented)
msg.o
```

DELETION OF DERIVED OBJECTS AND DATA CONTAINERS

Derived objects and their data containers can be deleted independently.

Removal of Data Containers

The standard *rm*(1) command works on a derived object in a natural way: the DO disappears from the view. The effect of physical data storage is as follows:

- If the DO's data container is in the view's private storage area, *rm* deletes that data container
- If the DO's data container is in a VOB storage pool, the data container remains unaffected.

In either case, the derived object in the VOB database is not deleted. The only change to the derived object is the decrementing of its reference count.

A build that overwrites an unshared DO works similarly. The operating system thinks it is merely overwriting an existing file; actually, the MVFS removes the old data container from the view's private storage area, and creates a new one there.

Removal of Derived Objects

The *rmdo* command removes a derived object from the VOB database. The effect of physical data storage is as follows:

- If the DO's data container is in the view's private storage area, it is *not* deleted. The data file continues to be visible, and will be listed by ClearCase's *ls* command with a `[no config record]` annotation.
- If the DO's data container is in a VOB storage pool, it is deleted along with the DO itself.

Scrubbing of Derived Objects and Data Containers

The *scrubber* utility removes derived objects from a VOB database and data containers from VOB storage pools. The *view_scrubber* utility removes data containers from a view's private storage area.

Degenerate Derived Objects

A derived object is "complete" if the DO itself, its data container, and its configuration record (CR) are all accessible. This may not be the case, however, given that these entities exist independently. A derived object can become incomplete, or *degenerate*, in these situations:

- **Data file deleted** — An unshared DO is removed with *rm* or by a target rebuild. The derived object continues to exist in the VOB database (with a zero reference count), but the data container no longer exists. Such DOs are usually ignored by *lsdo*, but can be listed with the `–zero` option. The *scrubber* utility deletes zero-referenced DOs.
- **DO deleted from VOB database** — An unshared DO is removed from its VOB database with *rmdo*. The data container continues to be visible:

```
% cleartool rmdo Vhelp.log
Removed derived object "Vhelp.log@@14-Sep.72783".

% cleartool ls Vhelp.log
Vhelp.log [no config record]
```

- **CR unavailable** — As described in the *config_record* manual page, a newly-created CR is stored in the view where its associated DO(s) were built. If that view becomes unavailable (for example, it is inadvertently destroyed or its host is temporarily down), the DO continues to exist in the VOB database, but operations that must access the CR will fail:

```
cleartool: Error: Unable to find view 'mercury:/viewstore/pink.vws'
          from albd: error detected by ClearCase subsystem
cleartool: Error: See albd_log on host mercury
cleartool: Error: Unable to contact View - error detected by ClearCase subsystem
```

DO VERSIONS

Subject to data-type restrictions imposed by the element type, you can *checkin* a derived object as a version of an element — a *DO version*. Other versions of such an element can also be, but need not be, derived objects. A DO version behaves both like a version and like a derived object:

- It has a *version-ID*, which can be used to reference it both as a VOB database object and as a data file.
- It can be assigned a version label, and can be referenced with that label.
- It has a configuration record, which can be used by *catcr* and *diffcr*. (These commands bypass the CR of a DO version, unless you use the `–ci` option.)
- It can be *winked-in* during a *clearmake* build, but only if it has been checked in at the same pathname where it was originally built.
- You can wink it in with a *winkin* command.
- It is listed by the *describe* command as a `derived object version`. (But it is not listed by the *lsdo* command at all.)

When you *checkout* a DO version, it is winked-in to your view. You can use a standard pathname to access the DO's file system data. But note that VOB-database access is handled in an idiosyncratic fashion:

- A standard pathname to the DO references the version in the VOB database from which the checkout was made:

```
% cleartool checkout -nc hello                          (wink–in derived object 'hello')
Checked out "hello" from version "/main/3".
% cleartool mklabel EXPER hello                         (using standard pathname ...
Created label "EXPER" on "hello" version "/main/3".     ... accesses version from which checkout was made)
```

- To access the *CHECKEDOUT* placeholder version, you must use an extended pathname:

```
% cleartool mklabel –replace EXPER hello@@/main/CHECKEDOUT
Moved label "EXPER" on "hello" from version "/main/3" to "/main/CHECKEDOUT".
```

If you process a checked-out DO version as described in "Manipulating Derived Objects with Standard Commands" above, ClearCase reverts to its usual handling of checked-out versions: a standard pathname references the placeholder version in the VOB database.

COMMANDS FOR WORKING WITH DERIVED OBJECTS

ClearCase includes commands for working with derived objects and their associated configuration records:

lsdo, describe	Lists a VOB's derived objects. *lsdo* does not list DO versions; *describe* does.
rmdo	Deletes derived objects and their data containers.
scrubber	Deletes derived objects and their data containers.
catcr	Lists the CR associated with a derived object.
diffcr	Lists the differences between two CRs.
mklabel -config mkattr -config	Attaches labels and attributes to the versions listed in a CR.
-config rule in config spec	Configures a view to select the versions listed in a CR.

UNIX HARD LINKS AND DERIVED OBJECTS

You cannot make a VOB hard link to a derived object. You *can* make one or more view-private hard links to a derived object, using the UNIX *ln* command, with these restrictions:

- The derived object must be visible in the view where the view-private hard link is to be created — that is, it must appear in a standard UNIX *ls* listing. (You can use the *winkin* command satisfy this requirement.)
- The pathname of the hard link must be within the same VOB as the original derived object.

All hard links to a derived object appear with the same *DO-ID* in a ClearCase *ls* listing; if there are two or more links in the same directory, they are all listed. For example:

```
% ln hello hw
% cleartool ls hello hw
hello@@19-May.19:15.232
hw@@19-May.19:15.232
```

All the hard links are equivalent to the *catcr* and *describe* commands, but *lsdo* "sees" a derived object only at its original name. Likewise, a derived object can be winked-in only at its original pathname.

SPECIAL CASE: If a hard link is created by the same build script as the derived object itself, then the hard link becomes an additional "original" name for the DO. *lsdo* will list the hard link, and *clearmake* will be able to perform wink-in at the hard link's pathname.

Each additional hard link increments a derived object's reference count. An `lsdo -l` listing includes the reference counts and the views in which the references exist. The `(2)` in this example shows that view *old.vws* has two references to *hello*:

```
% cleartool lsdo –long hello
08-Dec-92.12:06:19     Chuck Jackson (test user) (jackson.dvt@oxygen)
  create derived object "hello@@08-Dec.12:06.234"
  references: 2  => oxygen:/usr/vobstore/tut/old.vws (2)
```

SEE ALSO

cleartool subcommands: ls, lsdo, rmdo, catcr, diffcr
clearmake, clearaudit, config_record, makefile_ccase, scrubber, view_scrubber

NAME env_ccase – ClearCase environment variables

DESCRIPTION

This manual page describes the environment variables (EVs) used by ClearCase commands, programs, utilities, and software installation scripts. It includes both ClearCase-specific EVs and standard UNIX EVs that are particularly important for ClearCase usage.

Omitted are the EVs used by *triggers* and by the *find* commands; see the *mktrtype*, *find*, and *findmerge* manual pages for listings.

ATRIAHOME
: Installation directory for ClearCase software. Set this EV before running the *install_release* script to specify a non–standard installation location. On such hosts, user's shell startup scripts should use `$ATRIAHOME/bin` to specify the pathname of the ClearCase executables.

  Default: */usr/atria*.

ATRIA_LICENSE_HOST
: (HP-UX only) Hostname of the license server host. You must set this environment variable before running the *update*(1M) utility to install ClearCase for HP-UX.

  Default: none.

ATRIA_LINK_HOME
: (HP-UX only) Local pathname of the ClearCase installation area on the link host; for example, */net/neptune/usr/atria*. You must be set this environment variable before running the *update*(1M) utility to perform a LINK-mode installation of ClearCase for HP-UX.

  Default: none.

ATRIA_NO_BOLD
: A flag variable: if defined with a non–zero value, it suppresses generation of boldface characters in *cleartool* and *clearmake* output.

  Default: undefined.

ATRIA_RGY_HOST
: (HP-UX only) Hostname of the registry server host. You must set this environment variable before running the *update*(1M) utility to install ClearCase for HP-UX.

  Default: none.

ATRIA_RGY_REGION
: (HP-UX only) The network region of the local host. You must set this environment variable before running the *update*(1M) utility to install ClearCase for HP-UX.

  Default: none.

BITMAP_PATH
: Bitmap file search path. The icons that an *xclearcase* directory browser displays for file system objects are stored in bitmap files. It searches in directories on this colon-separated search path for such bitmap files.

Default: *home-directory*`/.bitmaps:${ATRIAHOME:-/usr/atria}/config/ui/bitmaps` . See also: ICON_PATH.

CLEARAUDIT_SHELL
The program that *clearaudit* runs in an audited shell. You must set this environment variable to the program's full pathname; for example, */bin/csh* or */usr/home/myscript*.

Default: *clearaudit* runs the program specified by the SHELL environment variable or, if SHELL is undefined, a Bourne shell (*/bin/sh*). See also: SHELL.

CLEARCASE_ABE_PN
The full pathname with which *clearmake* invokes the Audited Build Executor on a local or remote host during a distributed build.

Default: */bin/abe*.

CLEARCASE_AVOBS
A colon-separated list of full pathnames, each of which specifies a VOB storage directory. Certain commands use the value of this variable when they are invoked with the "all VOBS" option (`–avobs`).

All the VOBS that are currently mounted on the local host.

CLEARCASE_BLD_AUDIT_TMPDIR
The directory where *clearmake* and *clearaudit* create temporary build audit files. It must be a non-MVFS directory (that is, not below a VOB-tag).

Default: */tmp*.

CLEARCASE_BLD_CONC
Sets the concurrency level in a *clearmake* build. This EV takes the same values as the `–J` option. Specifying a `–J` option on the *clearmake* command line overrides the setting of this EV.

Default: none.

CLEARCASE_BLD_HOST_TYPE
The suffix of the *build hosts file*, which specifies hosts to be used for a distributed build (`clearmake -J`). The pathname of this file must have the form ***.bldhost.****suffix*, and must reside in your home directory. (Your home directory is determined by examining the password database.)

Default: none.

CLEARCASE_BLD_OPTIONS_SPECS
A colon-separated list of pathnames, each of which specifies a BOS file to be read by *clearmake*. You can use this EV instead of specifying BOS files on the *clearmake* command line with one or more `–A` options.

Default: undefined.

CLEARCASE_BLD_SHELL_FLAGS
Specifies *clearmake* command options to be passed to the subshell program that executes a build script command.

Default: `-e`.

CLEARCASE_BLD_UMASK
Sets the *umask*(1) value to be used for newly-created derived objects. It may be advisable to have this EV be more permissive than your standard umask — for example, CLEARCASE_BLD_UMASK = 2 where umask = 22. The reason is to create DOs that are more accessible than other files is *wink-in*: a winked-in file retains its original ownership and permissions. For example, when another user winks-in a file that you originally built, the file is still owned by you, is still a member of your principal group, and still has the permissions with which you created it. (NOTE: You can use the standard *chmod* command to change the permissions of a DO after you create it.)

CLEARCASE_BLD_UMASK can also be coded as a make macro.

Default: Same as current umask.

CLEARCASE_BLD_VERBOSITY
An integer that specifies the *clearmake* message logging level, from 0 (least verbose) to 2 (most verbose). Setting this EV to 1 is equivalent to specifying `clearmake -v`. Setting this EV to 2 is equivalent to specifying `clearmake -d`. Specifying `–v` or `–d` on the *clearmake* command line overrides the setting of this EV.

Default: 0.

CLEARCASE_DBG_GRP
Set this variable to a non-zero value to force *xclearcase* to print debugging information when executing button and menu commands in the graphical interface.

Default: none.

CLEARCASE_INCLUDE_CXX_RULES
In a makefile read by *clearmake*, expands to an `include` statement that reads in file *$(CLEARCASE_MAKE_CONFIG_DIR)/cxx.mk*. This file contains definitions that are used in a C++ environment to handle parameterized types (templates).

CLEARCASE_MAKE_COMPAT
clearmake's *make*-compatibility mode. This EV takes the same values as *clearmake*'s `–C` option. Specifying `–C` on the command line overrides the setting of this EV.

Default: none.

CLEARCASE_MAKE_CONFIG_DIR
In a makefile read by *clearmake*, expands to the full pathname of the *clearmake* configuration directory in the ClearCase installation area — typically */usr/atria/config/clearmake*.

CLEARCASE_MSG_PROTO
Enables one-way message forwarding between ClearCase and an interprogram messaging system (for example, *ToolTalk*). This feature allows ClearCase to notify the messaging system that an operation succeeded (for example, a checkout) without going through an encapsulator. For example, set this environment variable to `ToolTalk` to enable one-way message forwarding for the ToolTalk Broadcast Message Server. One-way message forwarding succeeds only if all programs involved have the same value for the DISPLAY environment variable.

Default: none. Supported values: `ToolTalk`, `SoftBench`. See also: DISPLAY, WINEDITOR.

CLEARCASE_PROFILE
The file containing your ClearCase user profile, which includes rules that determine the comment option default for one or more *cleartool* commands. This setting must be a full pathname.

Default: *.clearcase_profile* in your home directory.

CLEARCASE_REMOTE_USER
Specifies the Domain/OS user account to be used by *clearcvt_dsee* when importing a DSEE source library. The import script produced by *clearcvt_dsee* includes commands of the form:

rcp ${CLEARCASE_REMOTE_USER}*hostname***:/***path_to_dseelib***/***file*

For example, setting CLEARCASE_REMOTE_USER to `jones@` causes the *rcp* command to use **jones@***hostname* in its source address.

Default: none.

CLEARCASE_ROOT
The full pathname of the root directory of a *set view* process — a process created by the *setview* command. For example, the command `setview bugfix` creates a shell in which CLEARCASE_ROOT is set to `/view/bugfix`.

Default: not set in a process that was not created by *setview*.

CLEARCASE_TAB_SIZE
Specifies the tab width for output produced by *cleardiff*, *xcleardiff*, and source lines listed by the *cleartool* subcommand *annotate*.

Default: 8.

CLEARCASE_TRACE_TRIGGERS
A flag variable: if defined with a non-zero value, it causes all triggers to behave as if they were defined with the `–print` option when they fire.

Default: undefined.

CVT_REPLACE_COMM
A character string used by conversion scripts created by *clearcvt_unix* as the comment for `create version` event records.

Default: `made from unix file`.

CVT_TEMP_DIRECTORY
The directory where *clearcvt_unix* stores temporary files.

Default: */tmp*.

CVT_UPDATE
When set to any non-null string, forces a conversion script created by one of the ClearCase conversion utilities (for example, *clearcvt_rcs*) to consider source revisions individually as candidates for conversion. Default: not set, enabling a heuristic algorithm that can skip conversion of an entire source file.

Default: undefined.

DISPLAY
The X Window System display to use for ClearCase's GUI utilities (and all other X applications). If you are using an inter-program messaging system (for example, *ToolTalk*), all your tools must have the same DISPLAY value.

Default: undefined.

EDITOR
VISUAL
The pathname of a text editor. The *edcs* subcommand invokes the editor specified by the environment variable WINEDITOR (first choice), or VISUAL (second choice), or EDITOR (third choice). *xclearcase* invokes the editor specified by the environment variable WINEDITOR (first choice) or EDITOR (second choice).

Default: *vi*. See also: WINEDITOR.

GRP_PATH
A colon-separated list of files and directories to be searched for group files when you start *xclearcase*.
Default: *home-directory*`/.grp:${ATRIAHOME:-/usr/atria}/config/ui/grp`.

HOME
Not used — ClearCase programs determine your home directory by reading the password database, not by using this environment variable.

ICON_PATH
A colon-separated list of directories to be searched for *icon files*. *xclearcase* directory browsers use the bitmap images in such files as icons for file system objects.

Default: *home-directory*`/.icon:${ATRIAHOME:-/usr/atria}/config/ui/icon`
See also: BITMAP_PATH.

MAGIC_PATH
A colon-separated list of directories to be searched for *magic files*. Various ClearCase programs consult magic files to perform *file-typing* on file system objects.

Default: *home-directory*`/.magic:${ATRIAHOME:-/usr/atria}/config/magic`

MAKEFLAGS
Provides an alternative (or supplementary) mechanism for specifying *clearmake* command options. MAKEFLAGS can contain the same string of key letters used for command-line options, except that **f** and **r** are not allowed. Options on the *clearmake* command line override the setting of this environment variable if there is a conflict.

Default: none.

MANPATH
A colon-separated list of directories in which the UNIX *man*(1) command searches for manual pages. (The `cleartool man` command does not use MANPATH, but always searches in *${ATRIAHOME:-/usr/atria}/doc/man*.) Default: varies with operating system.

PATH
The standard UNIX program search path. To access ClearCase executables, change your search path to include directory *$ATRIAHOME/bin*.

Default: set by your shell program; typically modified in shell startup script.

SCHEMESEARCHPATH
A colon-separated list of directories to be searched for *scheme files*, which contain X Window System resource settings.
Default: */usr/lib/X11/Schemes:${ATRIAHOME:-/usr/atria}/config/ui/Schemes*.

SHELL
The default shell program to be run by various ClearCase commands and programs, including the *shell* and *setview* commands, and the *clearaudit* utility (if the environment variable CLEARAUDIT_SHELL is undefined).
Default: set by your shell program.

TERM
The kind of terminal for which output is to be prepared. Certain *cleartool* commands produce output that use special terminal capabilities. For example, *catcr* uses boldface to highlight information in a configuration record. To see bold characters in an *xterm*, set TERM to `xterm`, and provide a bold font with the X Toolkit option `-fb`, or with the X resource *xterm*boldFont*. To prevent the control characters that enable bolding from appearing in an *emacs* shell, set TERM to `emacs` in your *emacs* startup script, or set ATRIA_NO_BOLD.

Default: none; typically set in shell startup script.

WINEDITOR
An X Window System text editor application (for example, *xedit*(1)), which is invoked by *xclearcase* on a browser item. If WINEDITOR is undefined, *xclearcase* creates a terminal window, and runs the program specified by the EDITOR environment variable. If neither of these variables are defined, no editor is invoked.

Default: none.

SEE ALSO

cleartool subcommands: edcs, find, findmerge, man, mktrtype, setview, shell
cc.icon, cc.magic, clearaudit, clearcvt_unix, clearencap_sb, clearencap_tt, clearmake, profile_ccase, schemes, xcleardiff, xclearcase

NAME errorlogs_ccase – ClearCase error log files

SYNOPSIS

/usr/adm/atria/log/*logfile_name*

DESCRIPTION

ClearCase "log" files are located on each ClearCase host in the directory */usr/adm/atria/log*. Log files record error and status information from various ClearCase server programs and user programs, and include:

albd_log	used by the Location Broker Daemon (*albd_server*)
db_server_log	used by VOB database server, *db_server*
error_log	general-purpose error log, used by ClearCase user-programs, such as *cleartool*
event_scrubber_log	used by *event_scrubber* program
export_mvfs_log	used by *export_mvfs* program (not included in *ClearCase for OSF/1*)
install_log	used by *install_release* (ClearCase installation script)
lockmgr_log	used by *lockmgr* program
mntrpc_server_log	used by *mntrpc_server* program, which performs MVFS-file-system mounts requested by *cleartool* subcommand *mount*
promote_log	used by *promote_server*
scrubber_log	used by *scrubber* program
view_log	used by *view_server*
vob_log	used by *vob_server*
vob_scrubber_log	used by *vob_scrubber* program
vobrpc_server_log	used by *vobrpc_server*

Error log files are standard ASCII files. They can be edited, *grep*'ed, *cat*'ed, and so forth. A typical entry includes the date and time of the error, the software module in which the error occurred, the current user, and an error-specific message. The following is a typical example from the *view_log* file:

```
01/05/92 13:07:49 view_server(19314): Error: Set configuration
spec of .compiled_spec failed
```

As errors accumulate, the error log files grow. A ClearCase *crontab*(1M) script periodically renames error log files to *logfile_name****.old***, and creates empty "template" files in their place. See the *crontab_ccase* manual page for details.

SEE ALSO

crontab_ccase

NAME events_ccase – ClearCase operations and event records

DESCRIPTION

ClearCase creates an *event record* in the VOB database for nearly every operation that modifies the VOB. For example, if you create a new element, attach a version label, or lock the VOB, an event record marks the change.

Event records are attached to specific objects in VOB databases. Thus, each object (including the VOB object itself) accumulates a chronological *event history*, which you can display with the command *lshistory*.

In addition, you can:

- customize event history reports with `lshistory -fmt` (see the *fmt_ccase* manual page)
- scrub "minor" event records from the VOB database to save space; see the *vob_scrubber* manual page.
- assign *triggers* to many event-causing operations (*mkelem*, *checkout*, and *mklabel*, for example); see the *mktrtype* manual page
- change the comment stored with an event; see the *chevent* manual page.

Contents of an Event Record

An event record stores "who, what, when, where, and why?" information for various ClearCase operations:

obj-name	the object(s) affected
obj-kind	the kind of object (file element, branch, or label type, for example)
user-name	the user who changed the VOB database
host-name	the client host from which the VOB database was changed
operation	the ClearCase operation that caused the event (usually a *cleartool* command like *checkout* or *mklabel*)
date-time	when the operation occurred (reported relative to the local time zone).
event-kind	a description of the event, derived from a combination of the *operation* and *obj-kind* fields
comment	a text string — generated automatically by ClearCase, provided by *user-name*, or a combination of both

VOB Objects and Event Histories

The following kinds of VOB-database objects have event histories, which you can display with *lshistory*:

VOB
VOB storage pool
Element
Branch
Version
VOB symbolic link
Hyperlink
Derived Object (no creation event)
Replica
Type
    Attribute type
    Branch type
    Element type
    Hyperlink type
    Label type
    Trigger type
    Replica type

Each time an object from any of these categories is created, it begins its own event history with a creation event. (Derived objects are an exception; ClearCase stores a DO's creation time in its config record, not in an event record.) As time passes, some objects — VOBs and elements, in particular — can accumulate lengthy event histories.

Do not confuse type objects (created with *mkattype*, *mkbrtype*, *mkeltype*, *mkhltype*, *mklbtype*, and *mktrtype*) with the instances of those types (created with *mkattr*, *mkbranch*, *mkelem*, *mkhlink*, *mklabel*, and *mktrigger*). The type objects are VOB-database objects, with their own event histories. Individual branches, elements, and hyperlinks are also VOB-database objects. However, individual attributes, labels, and triggers are not and, therefore, do not have their own event histories. (Their create and delete events (*mkattr*/*rmattr*, *mklabel*/*rmlabel*, and *mktrigger*/*rmtrigger*) are recorded on the objects to which these meta-data items are attached.)

Operations that Cause Event Records to be Written

The following kinds of operations cause event records to be written to the VOB database.

- Create or import a new object.
- Destroy (remove) an object.
- Checkout a branch.
- Modify or delete version data.
- Modify a directory version's list of names.

- Attach or remove an attribute, label, hyperlink, or trigger.
- Lock or unlock an object.
- Change the name or definition of a type or storage pool.
- Change a branch or element's type.
- Change an element's storage pool.
- Change the protections for an element or derived object.

Table 4 lists event-causing operations as you might see them in *lshistory* output that has been formatted with the `-fmt` option's **%o** ("operation") specifier. Note that most operations correspond exactly to *cleartool* subcommands.

Table 4. Operations that Generate Event Records

Symbol	**Notes on the Operation or Its Event Records**
M	Causes a "minor" event (see `lshistory -minor`)
T	Can have a trigger (see *mktrtype*)
S	Resulting event records can be *scrubbed* (see *vob_scrubber*)
C	Generates a comment automatically (see "Comment Handling" in the *cleartool* manual page)

Operation that Generates the Event Record	**Notes (see key above)**	**Commands that Always Cause the Operation**	**Commands that May Cause the Operation**	**Object To Which the Event Record is Attached**
checkin	T	*checkin, mkelem, mkbranch*		newly created version
checkout	T	*checkout*	*findmerge, mkelem, mkbranch*	checked-out branch (event deleted automatically at checkin or uncheckout)
chpool	M S C	*chpool*		element
chtype	M T S C	*chtype*		element or branch
import			*clearcvt_**	imported element or type
lnname	M T S C	*ln, ln -s, mkelem, mkdir, mv*		directory version
lock	T S C	*lock*	*(various)*	locked object (type, pool, VOB, elem, or branch)
mkattr	M T S C	*mkattr*	*mkhlink*	element, branch, version, hlink, or VOB symlink
mkbranch	T	*mkbranch, mkelem*	*checkout*	new branch
mkelem	T C	*mkelem, mkdir*		new element

mkhlink	M	T	S	C	*mkhlink*	*merge, findmerge*	hyperlink object and "from" object, and for bidirectional hyperlinks, "to" object (unless cross-VOB hyperlink)
mklabel	M	T	S	C	*mklabel*		version
mkpool					*mkpool*		storage pool obj
mkslink		T			*ln -s*		directory version
mktrigger	M	T	S		*mktrigger*		element
mktype		T			*mk**type*		newly created type object
mkvob					*mkvob (causes numerous creation events)*		VOB
modpool	M		S	C	*mkpool -update*		storage pool
modtype	M		S	C	*mk**type -replace*		type object
protect	M		S	C	*protect*		element or DO
reformatvob					*reformatvob*		VOB
reserve	M	T			*reserve*		checked-out version
rmattr	M	T	S		*rmattr*		(see mkattr)
rmbranch		T	S	C	*rmbranch*		parent branch
rmelem		T	S	C	*rmelem*		VOB
rmhlink	M	T	S	C	*rmhlink, rmmerge*		"from" object, "to" object (unless cross-VOB, unidirectional), VOB
rmlabel	M	T	S		*rmlabel*		version
rmname	M	T	S	C	*rmname, rmelem, mv*		directory version(s)
rmpool			S	C	*rmpool*		VOB
rmtrigger	M	T	S		*rmtrigger*		element
rmtype		T	S	C	*rmtype*		VOB
rmver	M	T	S	C	*rmver*		
rnpool	M			C	*rnpool*		storage pool
rntype	M	T		C	*rntype*		type object
unlock		T	S		*unlock*	*(various)*	unlocked object
unreserve	M	T			*unreserve*		checked-out version

Operations and Triggers

Each of the following "super-operations" represents a group of the above event-causing operations. See *mktrtype* for information on how to use the following keywords to write triggers for groups of operations.

MODIFY_TYPE MODIFY_DATA
MODIFY_ELEM MODIFY_MD

Table 4 omits the triggerable operations *uncheckout* and *chevent*, as these operations do not cause event records to be stored in the VOB database.

Event Visibility

This section describes where, directly or indirectly, you might encounter event record contents. The following commands include event history information in their output, which can be formatted with the –fmt option:

Command	Relevant VOB Objects	Event Information Reported
describe	all	creation or checkout event
lscheckout	branch	checkout event
lshistory	all	controlled by command options
lslock	VOB, element, branch, pool, type object	lock event
lspool	pools	creation event
lstype -long	type objects	creation event

Comments and Event Records. The set of ClearCase commands named in the above table of event-causing operations matches almost exactly the set of commands that accept user comments as input. (*reformatvob*, which takes no comment, is the only exception.) When you supply comments to a ClearCase command, your comment becomes part of an event record.

Some *cleartool* commands create a comment automatically, even if you do not provide one. These *generated comments* describe the operation in general terms, such as “modify meta-data” or “create dir element”. User comments, if any, are appended to generated comments. For a complete description of comment-related command options and comment processing, see “Comment Handling” in the *cleartool* manual page.

SEE ALSO

cleartool subcommands: chevent, lshistory, mktrtype
fmt_ccase, vob_scrubber, "Comment Handling" in cleartool manual page

NAME export_mvfs – export and unexport VOBs to NFS clients (non-ClearCase access)

SYNOPSIS

/usr/etc/export_mvfs [**–a**] [**–v**] [**–i**] [**–u**] [**–o** *options*] [*pname*]

- SunOS-5 only: pathname is **/usr/atria/etc/export_mvfs**

DESCRIPTION

No 'export_mvfs' utility is supplied with ClearCase for OSF/1. See the 'exports_osf1' manual page for details on exporting VOBs on OSF/1 systems.

export_mvfs is the ClearCase counterpart of the *exportfs*(1M) command (SunOS–4, IRIX–5, HPUX–9) or *share*(1M) command (SunOS-5) for file systems of type MVFS. This utility enables *non-ClearCase access*; it makes a local VOB available for mounting over the network by hosts on which ClearCase is not installed.

export_mvfs is normally invoked at system startup by the ClearCase startup script. It uses information in file */etc/exports.mvfs* to export one or more VOBs through view–extended pathnames. You can run *export_mvfs* manually to export or unexport an individual VOB.

With no options or arguments, *export_mvfs* lists the VOBs currently exported by the host.

OPTIONS AND ARGUMENTS

–a (all) Exports all pathnames listed in */etc/exports.mvfs*; with –u, unexports all currently exported VOBs.

–v (verbose) Displays each pathname as it is exported or unexported.

–i Ignores the options in */etc/exports.mvfs*. By default, *export_mvfs* consults */etc/exports.mvfs* for the options associated with each pathname to be exported.

–u Unexports the specified pathnames; with –a, unexports all currently exported VOBs.

–o *options* A comma–separated list of optional characteristics for the pathnames being exported. See the *exports_ccase* manual page for a list of supported options.

pathname A view-extended pathname to the VOB-tag (mount point) of the VOB to be exported.

FILES

/etc/exports.mvfs
/usr/etc/export_mvfs (all but SunOS-5)
/usr/atria/etc/export_mvfs (SunOS-5 only)

SEE ALSO

exports_ccase, filesys_ccase, init_ccase
exportfs(1M), exports(4), share(1M)

NAME exports_ccase – list of VOBs to be accessed by non-ClearCase hosts

DESCRIPTION

ClearCase VOBs can be *exported* through particular views, for access by non-ClearCase hosts in the local network. The mechanisms for exporting VOBs are architecture-specific. Consult the following manual pages:

SunOS-4	exports_sun4
SunOS-5	exports_sun5
HPUX-9	exports_hpx9
IRIX-5	exports_irx5
OSF/1 V2	exports_osf1

NAME exports__arch_ – list of VOBs to be accessed by non-ClearCase hosts (exporting from HPUX-9)

SYNOPSIS

- Exports table entry:

  VOB-tag [*options*] [*netgroup*] [*hostname* …]

- Standard options:

  ro, **rw**, **anon**, **root**, **access**
  Default options: **rw**, **anon=nobody**

DESCRIPTION

A host that has not installed ClearCase can still access any VOB, using NFS. Several steps are involved in providing such *non-ClearCase access* to a VOB:

- A ClearCase *client host* — one whose kernel includes the MVFS — activates (mounts) the VOB.
- The host starts an *export view*, through which the VOB will be accessed by non-ClearCase hosts.
- The host uses a ClearCase-specific exports file to export a view-extended pathname to the VOB-tag (mount point) — for example, */view/exp_vu/vobs/proj*.
- One or more non-ClearCase hosts in the network perform an NFS mount of the exported pathname.

The file */etc/exports.mvfs* is the ClearCase counterpart of the standard UNIX */etc/exports* file. (You cannot use */etc/exports* to export a VOB.) At system startup, the ClearCase startup script invokes the *export_mvfs* utility to process the entries in */etc/exports.mvfs*. Each entry in this file enables access to one VOB by non-ClearCase hosts.

A VOB-export entry has the format shown in the "Synopsis" section, above. Export options, hostnames, and comments are processed as described in the *exports*(4) manual page. If you use an NFS "soft mount" to access the VOB, allow enough time for successful mounting by:

- setting the timeout (`timeo`) parameter to a value ≥ 30
- setting the NFS retransmission (`retrans`) parameter to a value ≥ 5

To improve performance, use the `–access` option to restrict the export to a particular set of hosts and/or netgroups (see example below).

RESTRICTIONS

When setting up non-ClearCase access, you must observe these restrictions:

- Any VOB to be exported at system startup must be listed in the ClearCase storage registry as a *public* VOB.
- The *VOB-tag* (VOB mount point) must be a specified as a view-extended pathname. Examples:

```
/view/gamma/vobs/proj
/view/alpha/vobs/vega     rw=mercury:venus:jupiter
```

- The view storage directory must be located on the local host.

We strongly recommend that you observe these additional restrictions:

- The view storage directory and VOB storage directory involved in the export reside on the same host.
- The data storage for both the view and the VOB must be local — no remote storage pools for the VOB; no remote private storage area for the view.

If you do not wish to (or cannot) observe these additional restrictions, consult "Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*.

EXAMPLES

- A ClearCase host, *saturn*, wishes to export a VOB mounted at */vobs/proj*, as seen through view *beta*. This line exports the VOB to all hosts in the network:

```
/view/beta/vobs/proj
```

A non-ClearCase host soft-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /ccase_vobs/proj  nfs  rw,noauto,soft,timeo=300,retrans=10 0 0
```

Another host hard-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /vobs/proj  nfs rw,hard  0  0
```

- Export a VOB to netgroup *pcgroup*, and also to individual host *newton*. In addition, specify the export option `rw`.

```
/view/beta/vobs/proj  -rw,access=pcgroup:newton
```

NOTES

The ClearCase installation procedure creates a template */etc/exports.mvfs*, all of whose lines are commented out.

FILES

/etc/exports.mvfs
/etc/rc.atria

SEE ALSO

cleartool, exports_ccase, filesys_ccase, export_mvfs, exports(4), fstab(4), netgroup(4)
"Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*

NAME exports__arch_ – list of VOBs to be accessed by non-ClearCase hosts (exporting from IRIX-5)

SYNOPSIS

- Exports table entry:

  VOB-tag [*options*] [*netgroup*] [*hostname* …]

- Standard options:

  ro, **rw**, **anon**, **root**, **access**
  Default options: **rw**, **anon=nobody**

DESCRIPTION

A host that has not installed ClearCase can still access any VOB, using NFS. Several steps are involved in providing such *non-ClearCase access* to a VOB:

- A ClearCase *client host* — one whose kernel includes the MVFS — activates (mounts) the VOB.
- The host starts an *export view*, through which the VOB will be accessed by non-ClearCase hosts.
- The host uses a ClearCase-specific exports file to export a view-extended pathname to the VOB-tag (mount point) — for example, */view/exp_vu/vobs/proj*.
- One or more non-ClearCase hosts in the network perform an NFS mount of the exported pathname.

The file */etc/exports.mvfs* is the ClearCase counterpart of the standard UNIX */etc/exports* file. (You cannot use */etc/exports* to export a VOB.) At system startup, the ClearCase startup script invokes the *export_mvfs* utility to process the entries in */etc/exports.mvfs*. Each entry in this file enables access to one VOB by non-ClearCase hosts.

A VOB-export entry has the format shown in the "Synopsis" section, above. Export options, hostnames, and comments are processed as described in the *exports*(4) manual page. If you use an NFS "soft mount" to access the VOB, allow enough time for successful mounting by:

- setting the timeout (`timeo`) parameter to a value ≥ 30
- setting the NFS retransmission (`retrans`) parameter to a value ≥ 5

To improve performance, use the `–access` option to restrict the export to a particular set of hosts and/or netgroups (see example below).

RESTRICTIONS

When setting up non-ClearCase access, you must observe these restrictions:

- Any VOB to be exported at system startup must be listed in the ClearCase storage registry as a *public* VOB.
- The *VOB-tag* (VOB mount point) must be a specified as a view-extended pathname. Examples:

```
/view/gamma/vobs/proj
/view/alpha/vobs/vega     rw=mercury:venus:jupiter
```

- The view storage directory must be located on the local host.

We strongly recommend that you observe these additional restrictions:

- The view storage directory and VOB storage directory involved in the export reside on the same host.
- The data storage for both the view and the VOB must be local — no remote storage pools for the VOB; no remote private storage area for the view.

If you do not wish to (or cannot) observe these additional restrictions, consult "Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*.

EXAMPLES

- A ClearCase host, *saturn*, wishes to export a VOB mounted at */vobs/proj*, as seen through view *beta*. This line exports the VOB to all hosts in the network:

```
/view/beta/vobs/proj
```

A non-ClearCase host soft-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /ccase_vobs/proj  nfs  rw,noauto,soft,timeo=300,retrans=10 0 0
```

Another host hard-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /vobs/proj  nfs rw,hard  0  0
```

- Export a VOB to netgroup *pcgroup*, and also to individual host *newton*. In addition, specify the export option `rw`.

```
/view/beta/vobs/proj  -rw,access=pcgroup:newton
```

NOTES

The ClearCase installation procedure creates a template */etc/exports.mvfs*, all of whose lines are commented out.

FILES

/etc/exports.mvfs
/etc/init.d/atria

SEE ALSO

cleartool, exports_ccase, filesys_ccase, export_mvfs, exports(4), fstab(4), netgroup(4)
"Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*

NAME exports__arch_ – list of VOBs to be accessed by non-ClearCase hosts (exporting from OSF/1)

SYNOPSIS

- Exports table entry:

  VOB-tag [*options*] [*netgroup*] [*hostname* …]

- Standard options:

  ro, **rw**, **anon**, **root**, **access**
  Default options: **rw**, **anon=nobody**

DESCRIPTION

A host that has not installed ClearCase can still access any VOB, using NFS. Several steps are involved in providing such *non-ClearCase access* to a VOB:

- A ClearCase *client host* — one whose kernel includes the MVFS — activates (mounts) the VOB.
- The host starts an *export view*, through which the VOB will be accessed by non-ClearCase hosts.
- The host uses the standard exports file to export a view-extended pathname to the VOB-tag (mount point) — for example, */view/exp_vu/vobs/proj*.
- One or more non-ClearCase hosts in the network perform an NFS mount of the exported pathname.

VOBs are exported to non-ClearCase hosts using the standard */etc/exports* file. At system startup, the ClearCase startup script invokes *showmount*(8), which causes *mountd*(8) to export the entries in */etc/exports*. Each ClearCase-related entry in this file enables access to one VOB by non-ClearCase hosts.

NOTE: Exports sometimes fail due to a timing error. You can enter the command `touch /etc/exports` (as *root*) to re-export the VOBs.

A VOB-export entry has the format shown in the "Synopsis" section, above. Export options, hostnames, and comments are processed as described in the *exports*(4) manual page. If you use an NFS "soft mount" to access the VOB, allow enough time for successful mounting by:

- setting the timeout (`timeo`) parameter to a value ≥ 30
- setting the NFS retransmission (`retrans`) parameter to a value ≥ 5

To improve performance, use the `–access` option to restrict the export to a particular set of hosts and/or netgroups (see example below).

Caution

VOBs exported through this mechanism should all be *public* VOBs. The ClearCase startup script mounts all public VOBs automatically. If a VOB is not mounted at the time the export operation is attempted, the NFS mount daemon exits and the export fails.

RESTRICTIONS

When setting up non-ClearCase access, you must observe these restrictions:

- Any VOB to be exported at system startup must be listed in the ClearCase storage registry as a *public* VOB.
- The *VOB-tag* (VOB mount point) must be a specified as a view-extended pathname. Examples:

```
/view/gamma/vobs/proj
/view/alpha/vobs/vega     rw=mercury:venus:jupiter
```

- The view storage directory must be located on the local host.

We strongly recommend that you observe these additional restrictions:

- The view storage directory and VOB storage directory involved in the export reside on the same host.
- The data storage for both the view and the VOB must be local — no remote storage pools for the VOB; no remote private storage area for the view.

If you do not wish to (or cannot) observe these additional restrictions, consult "Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*.

EXAMPLES

- A ClearCase host, *saturn*, wishes to export a VOB mounted at */vobs/proj*, as seen through view *beta*. This line exports the VOB to all hosts in the network:

```
/view/beta/vobs/proj
```

A non-ClearCase host soft-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /ccase_vobs/proj  nfs  rw,noauto,soft,timeo=300,retrans=10 0 0
```

Another host hard-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /vobs/proj  nfs rw,hard  0  0
```

- Export a VOB to netgroup *pcgroup*, and also to individual host *newton*. In addition, specify the export option `rw`.

```
/view/beta/vobs/proj  -rw,access=pcgroup:newton
```

FILES

/sbin/init.d/atria

SEE ALSO

cleartool, exports_ccase, filesys_ccase, showmount(8), mountd(8), exports(4), fstab(4), netgroup(4)
"Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*

NAME exports__arch_ – list of VOBs to be accessed by non-ClearCase hosts (exporting from SunOS-4)

SYNOPSIS

- Exports table entry:

  VOB-tag [*options*] [*netgroup*] [*hostname* …]

- Standard options:

  ro, **rw**, **anon**, **root**, **access**
  Default options: **rw**, **anon=nobody**

DESCRIPTION

A host that has not installed ClearCase can still access any VOB, using NFS. Several steps are involved in providing such *non-ClearCase access* to a VOB:

- A ClearCase *client host* — one whose kernel includes the MVFS — activates (mounts) the VOB.
- The host starts an *export view*, through which the VOB will be accessed by non-ClearCase hosts.
- The host uses a ClearCase-specific exports file to export a view-extended pathname to the VOB-tag (mount point) — for example, */view/exp_vu/vobs/proj*.
- One or more non-ClearCase hosts in the network perform an NFS mount of the exported pathname.

The file */etc/exports.mvfs* is the ClearCase counterpart of the standard UNIX */etc/exports* file. (You cannot use */etc/exports* to export a VOB.) At system startup, the ClearCase startup script invokes the *export_mvfs* utility to process the entries in */etc/exports.mvfs*. Each entry in this file enables access to one VOB by non-ClearCase hosts.

A VOB-export entry has the format shown in the "Synopsis" section, above. Export options, hostnames, and comments are processed as described in the *exports*(4) manual page. If you use an NFS "soft mount" to access the VOB, allow enough time for successful mounting by:

- setting the timeout (`timeo`) parameter to a value ≥ 30
- setting the NFS retransmission (`retrans`) parameter to a value ≥ 5

To improve performance, use the `–access` option to restrict the export to a particular set of hosts and/or netgroups (see example below).

RESTRICTIONS

When setting up non-ClearCase access, you must observe these restrictions:

- Any VOB to be exported at system startup must be listed in the ClearCase storage registry as a *public* VOB.
- The *VOB-tag* (VOB mount point) must be a specified as a view-extended pathname. Examples:

```
/view/gamma/vobs/proj
/view/alpha/vobs/vega     rw=mercury:venus:jupiter
```

- The view storage directory must be located on the local host.

We strongly recommend that you observe these additional restrictions:

- The view storage directory and VOB storage directory involved in the export reside on the same host.
- The data storage for both the view and the VOB must be local — no remote storage pools for the VOB; no remote private storage area for the view.

If you do not wish to (or cannot) observe these additional restrictions, consult "Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*.

EXAMPLES

- A ClearCase host, *saturn*, wishes to export a VOB mounted at */vobs/proj*, as seen through view *beta*. This line exports the VOB to all hosts in the network:

```
/view/beta/vobs/proj
```

A non-ClearCase host soft-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /ccase_vobs/proj  nfs  rw,noauto,soft,timeo=300,retrans=10 0 0
```

Another host hard-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /vobs/proj  nfs rw,hard  0  0
```

- Export a VOB to netgroup *pcgroup*, and also to individual host *newton*. In addition, specify the export option `rw`.

```
/view/beta/vobs/proj  -rw,access=pcgroup:newton
```

NOTES

The ClearCase installation procedure creates a template */etc/exports.mvfs*, all of whose lines are commented out.

FILES

/etc/exports.mvfs
/etc/rc.atria

SEE ALSO

cleartool, exports_ccase, filesys_ccase, export_mvfs, exports(4), fstab(4), netgroup(4)
"Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*

NAME exports__arch_ – list of VOBs to be accessed by non-ClearCase hosts (exporting from SunOS-5)

SYNOPSIS

- Exports table entry:

  VOB-mount-point [**ro**] [**ro=***client*[**:***client*]…] [**rw**] [**rw**=*client*[**:***client*]…]
  [**anon=***uid*] [**root=***host*[**:***host*]…]

- Standard options:

  ro, **rw**, **anon**, **root**, **access**
  Default options: **rw**, **anon=nobody**

DESCRIPTION

A host that has not installed ClearCase can still access any VOB, using NFS. Several steps are involved in providing such *non-ClearCase access* to a VOB:

- A ClearCase *client host* — one whose kernel includes the MVFS — activates (mounts) the VOB.
- The host starts an *export view*, through which the VOB will be accessed by non-ClearCase hosts.
- The host uses a ClearCase-specific exports file to export a view-extended pathname to the VOB-tag (mount point) — for example, */view/exp_vu/vobs/proj*.
- One or more non-ClearCase hosts in the network perform an NFS mount of the exported pathname.

The file */etc/exports.mvfs* is the ClearCase counterpart of the standard UNIX */etc/dfs/dfstab* file. (You cannot use */etc/dfs/dfstab* to export a VOB.) At system startup, the ClearCase startup script invokes the *export_mvfs* utility to process the entries in */etc/exports.mvfs*. Each entry in this file enables access to one VOB by non-ClearCase hosts.

A VOB-export entry has the format shown in the "Synopsis" section, above. Export options, hostnames, and comments are processed as described in the *dfstab*(4) manual page. If you use an NFS "soft mount" to access the VOB, allow enough time for successful mounting by:

- setting the timeout (`timeo`) parameter to a value ≥ 30
- setting the NFS retransmission (`retrans`) parameter to a value ≥ 5

To improve performance, use the `–access` option to restrict the export to a particular set of hosts and/or netgroups (see example below).

RESTRICTIONS

When setting up non-ClearCase access, you must observe these restrictions:

- Any VOB to be exported at system startup must be listed in the ClearCase storage registry as a *public* VOB.
- The *VOB-tag* (VOB mount point) must be a specified as a view-extended pathname. Examples:

```
/view/gamma/vobs/proj
/view/alpha/vobs/vega     rw=mercury:venus:jupiter
```

- The view storage directory must be located on the local host.

We strongly recommend that you observe these additional restrictions:

- The view storage directory and VOB storage directory involved in the export reside on the same host.
- The data storage for both the view and the VOB must be local — no remote storage pools for the VOB; no remote private storage area for the view.

If you do not wish to (or cannot) observe these additional restrictions, consult "Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*.

EXAMPLES

- A ClearCase host, *saturn*, wishes to export a VOB mounted at */vobs/proj*, as seen through view *beta*. This line exports the VOB to all hosts in the network:

```
/view/beta/vobs/proj
```

A non-ClearCase host soft-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /ccase_vobs/proj  nfs  rw,noauto,soft,timeo=300,retrans=10 0 0
```

Another host hard-mounts the VOB with this file system table entry:

```
saturn:/view/beta/vobs/proj  /vobs/proj  nfs rw,hard  0  0
```

- Export a VOB to netgroup *pcgroup*, and also to individual host *newton*. In addition, specify the export option `rw`.

```
/view/beta/vobs/proj  -rw,access=pcgroup:newton
```

NOTES

The ClearCase installation procedure creates a template */etc/exports.mvfs*, all of whose lines are commented out.

FILES

/etc/exports.mvfs
/etc/init.d/atria

SEE ALSO

cleartool, exports_ccase, filesys_ccase, export_mvfs, dfstab(4), vfstab(4), netgroup(4)
"Setting Up an Export View for Non-ClearCase Access" in the *ClearCase Administrator´s Manual*

NAME filesys_ccase – file system table entries for VOBs: fstab.mvfs

DESCRIPTION

Each VOB is mounted as a file system of type MVFS. A host's viewroot directory is also mounted as a file system of type MVFS. The file system tables that specify these mounts are architecture-specific. Consult the following manual pages:

SunOS-4	filesys_sun4, mount_sun4
SunOS-5	filesys_sun5, mount_sun5
HPUX-9	filesys_hpx9, mount_hpx9
IRIX-5	filesys_irx5, mount_irx5
OSF/1 V2	filesys_osf1, mount_osf1

NAME filesys_hpx9 – file system table entries for VOBs: fstab.mvfs (HPUX-9)

SYNOPSIS

- VOB mount entry:

  VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

- Non-standard file system type:

  mvfs

- VOB mount options:

  ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **timeo**, **retrans**,
  acregmin, **acregmax**, **acdirmin**, **acdirmax**

- "Obsolete" script for automatically mounting VOBs (*root* only):

  /usr/atria/etc/clearcase_domounts { **file** *filesys-table-pname* | **nis** *NIS-map-name* }

DESCRIPTION

The functionality described in this manual page has been rendered obsolete by the ClearCase VOB registry and the 'cleartool mount' subcommand. Support for this functionality is included in this release, but will be withdrawn in a future release.

To enable access by developers, a *versioned object base* (VOB) must be mounted on a directory as a file system of type *MVFS* (*multiversion file system*). VOB mount entries can be placed in your host's standard file system table, */etc/checklist*, or in the ClearCase-specific table, */etc/fstab.mvfs*. After a VOB has been mounted on a host, it can be accessed through any *view* that is active on that host.

NOTE: The *fstab.mvfs* file is known only to ClearCase software. It can be processed with the script */usr/atria/etc/clearcase_domounts*, which is described below. The standard UNIX utilities *mount* and *umount* do not process this file at all.

Viewroot Directory

Each host running ClearCase client software must also mount an MVFS file system called the *viewroot directory* (standard name: */view*). The ClearCase startup script performs this task — see the *init_ccase* manual page. There is no way to mount the viewroot directory through an entry in *fstab.mvfs*.

VOB MOUNT ENTRIES

The format of an entry for mounting a VOB closely resembles that for mounting a local file system:

VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

VOB-storage-pathname

Specifies an existing VOB storage directory, (created with *mkvob*). You cannot specify a subdirectory within the VOB; nor can you specify an ancestor directory (such as */usr/src*). This must be a local pathname; a *hostname*:*pathname* specification is invalid. See "Mounting a Remote VOB" below.

VOB-mount-point
Specifies the mount point, which may be any directory. (Typically, it is an empty one.) It must not be the same as the *VOB-storage-pathname* specification.

mvfs The file system type must be **mvfs** (*multiversion file system*).

options All standard UNIX file system table options are supported. See the standard *mount*(1M) manual page for a listing of the default settings; and see "Notes on Mount Options" below.

frequency
pass These parameters have their standard meanings.

Notes on Mount Options

A VOB can be mounted read-write (`rw` option), or read-only (`ro` option).

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` option in the VOB's file system table entry.

Using the `soft` option (recommended) causes ClearCase to return an error if a *view_server* process accessing the VOB goes down and cannot be restarted. Specifying `hard` causes the system to hang in such circumstances.

For VOB mounts, if you don't specify a timeout or retransmission option, a default value is used:

timeo 5 seconds
retrans 7 retries

MOUNTING A REMOTE VOB

A host running ClearCase software can access a VOB that is stored on another ClearCase host in the network. The file system in which the VOB storage directory physically resides must be NFS-mounted on the local host; it does not matter if the NFS mount occurs before or after the associated MVFS mount. Typically, the NFS mount is handled by a file system table entry; however, it could also be auto-mounted.

As with all NFS mounts, the file system containing the VOB storage directory must be listed in its host's */etc/exports* file (see *exports*(4)).

Example:

A VOB storage directory *proj.vbs* is created in the file system */usr/src.export* on remote host *venus*. The following file system table entry on the local host, *saturn*, provides a path to the VOB storage directory:

```
venus:/usr/src.export  /usr/src.import  nfs  rw,soft
```

This entry mounts the VOB as a type-MVFS file system:

```
/usr/src.import/proj.vbs  /usr/src/proj  mvfs rw,soft 0 0
```

CENTRAL ADMINISTRATION OF VOB MOUNT POINTS

An efficient way for a group of hosts in a network to share a set of VOBs is to use a Network Information Services (NIS) map. When it reads the ClearCase-specific file system table *fstab.mvfs*, the *mount_mvfs* utility recognizes a line in the following form as a reference to an NIS map:

+*map-name*

In effect, the entire contents of the map replaces the line. Such NIS map references can occur any number of times in the *fstab.mvfs* file. Moreover, such references can be nested — a map can reference other maps, using lines in the same form.

Example:

```
#
# vob                  mnt                FS      FS
# path                 path               type    options
#
/usr/src/project.vbs   /usr/src/project   mvfs     rw,soft
#
# Use the NIS map "clearcase_vobs" for all other VOB mounts
#
+clearcase_vobs
```

NIS Map Format

Lines in the NIS map have a slightly different format from lines in the file system table. The first two fields are reversed: in the NIS map, the first field specifies the mount point, and the second field specifies the VOB storage area.

The following example shows both the reverse field order and the use of a nested NIS map:

```
#
# mount pt    vob storage                      FS    FS
# path        path                             type  options
#
/vobs/public  /net/saturn/usr/vobstore/public  mvfs   rw,soft
/vobs/design  /net/saturn/usr/vobstore/design  mvfs   rw,soft
#
# NIS sub-map
#
+current_test_vobs
```

NOTE: The viewroot mount should *not* be specified in an NIS map. This restriction is for reliability: when NIS services are unavailable, you will still want to be able to use views on your host. Be sure to place the viewroot mount entry in your host's standard file system table.

THE CLEARCASE_DOMOUNTS SCRIPT

The *clearcase_domounts* script processes VOB-mount entries, either in a specified ASCII file or in a specified NIS map:

- Use the argument `file` to specify the pathname of an ASCII file
- Use the argument `NIS` to specify the name of an NIS map.

The file or NIS map must be in the format described in section "VOB Mount Entries" above.

The *clearcase_domounts* script uses the "new awk" program, and can use environment variable NAWK to determine the location of this program. If there is no program named *nawk* on your search path, use the EV to specify its full pathname. For example:

```
env NAWK=/usr/bin/awk /usr/atria/etc/clearcase_domounts file /etc/fstab.mvfs
```

EXAMPLES

- Soft-mounted VOB entry in file system table:

```
/usr/src/lib/vob  /vobs/lib   mvfs   rw,soft 0 0
```

- Viewroot mount command with non-default extended naming symbol:

```
mount -t mvfs -o viewroot,xnsuffix=%% /view /view
```

FILES

/etc/checklist
/etc/fstab.mvfs
/etc/rc.atria

SEE ALSO

cleartool subcommands: mktag, mkview, mkvob, mount, setview, startview, umount
albd_server, init_ccase, exports_ccase, mount(1M), mount(1M)

NAME filesys_irx5 – file system table entries for VOBs: fstab.mvfs (IRIX-5)

SYNOPSIS

- VOB mount entry:

  VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

- Non-standard file system type:

  mvfs

- VOB mount options:

  ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **timeo**, **retrans**, **acregmin**, **acregmax**, **acdirmin**, **acdirmax**

- "Obsolete" script for automatically mounting VOBs (*root* only):

  /usr/atria/etc/clearcase_domounts { **file** *filesys-table-pname* | **nis** *NIS-map-name* }

DESCRIPTION

The functionality described in this manual page has been rendered obsolete by the ClearCase VOB registry and the 'cleartool mount' subcommand. Support for this functionality is included in this release, but will be withdrawn in a future release.

To enable access by developers, a *versioned object base* (VOB) must be mounted on a directory as a file system of type *MVFS* (*multiversion file system*). VOB mount entries can be placed in your host's standard file system table, */etc/fstab*, or in the ClearCase-specific table, */etc/fstab.mvfs*. After a VOB has been mounted on a host, it can be accessed through any *view* that is active on that host.

NOTE: The *fstab.mvfs* file is known only to ClearCase software. It can be processed with the script */usr/atria/etc/clearcase_domounts*, which is described below. The standard UNIX utilities *mount*, *umount*, and *mountall* do not process this file at all.

Viewroot Directory

Each host running ClearCase client software must also mount an MVFS file system called the *viewroot directory* (standard name: */view*). The ClearCase startup script performs this task — see the *init_ccase* manual page. There is no way to mount the viewroot directory through an entry in *fstab.mvfs*.

VOB MOUNT ENTRIES

The format of an entry for mounting a VOB closely resembles that for mounting a local file system:

VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

VOB-storage-pathname

Specifies an existing VOB storage directory, (created with *mkvob*). You cannot specify a subdirectory within the VOB; nor can you specify an ancestor directory (such as */usr/src*). This must be a local pathname; a *hostname*:*pathname* specification is invalid. See "Mounting a Remote VOB" below.

VOB-mount-point
Specifies the mount point, which may be any directory. (Typically, it is an empty one.) It must not be the same as the *VOB-storage-pathname* specification.

mvfs The file system type must be **mvfs** (*multiversion file system*).

options All standard UNIX file system table options are supported. See the standard *mount*(1M) manual page for a listing of the default settings; and see "Notes on Mount Options" below.

frequency
pass These parameters have their standard meanings.

Notes on Mount Options

A VOB can be mounted read-write (`rw` option), or read-only (`ro` option).

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` option in the VOB's file system table entry.

Using the `soft` option (recommended) causes ClearCase to return an error if a *view_server* process accessing the VOB goes down and cannot be restarted. Specifying `hard` causes the system to hang in such circumstances.

For VOB mounts, if you don't specify a timeout or retransmission option, a default value is used:

timeo 5 seconds
retrans 7 retries

MOUNTING A REMOTE VOB

A host running ClearCase software can access a VOB that is stored on another ClearCase host in the network. The file system in which the VOB storage directory physically resides must be NFS-mounted on the local host; it does not matter if the NFS mount occurs before or after the associated MVFS mount. Typically, the NFS mount is handled by a file system table entry; however, it could also be auto-mounted.

As with all NFS mounts, the file system containing the VOB storage directory must be listed in its host's */etc/exports* file (see *exports*(4)).

Example:

A VOB storage directory *proj.vbs* is created in the file system */usr/src.export* on remote host *venus*. The following file system table entry on the local host, *saturn*, provides a path to the VOB storage directory:

```
venus:/usr/src.export  /usr/src.import  nfs  rw,soft
```

This entry mounts the VOB as a type-MVFS file system:

```
/usr/src.import/proj.vbs  /usr/src/proj  mvfs rw,soft 0 0
```

CENTRAL ADMINISTRATION OF VOB MOUNT POINTS

An efficient way for a group of hosts in a network to share a set of VOBs is to use a Network Information Services (NIS) map. When it reads the ClearCase-specific file system table *fstab.mvfs*, the *mount_mvfs* utility recognizes a line in the following form as a reference to an NIS map:

+*map-name*

In effect, the entire contents of the map replaces the line. Such NIS map references can occur any number of times in the *fstab.mvfs* file. Moreover, such references can be nested — a map can reference other maps, using lines in the same form.

Example:

```
#
# vob                  mnt                FS      FS
# path                 path               type    options
#
/usr/src/project.vbs   /usr/src/project   mvfs     rw,soft
#
# Use the NIS map "clearcase_vobs" for all other VOB mounts
#
+clearcase_vobs
```

NIS Map Format

Lines in the NIS map have a slightly different format from lines in the file system table. The first two fields are reversed: in the NIS map, the first field specifies the mount point, and the second field specifies the VOB storage area.

The following example shows both the reverse field order and the use of a nested NIS map:

```
#
# mount pt    vob storage                      FS    FS
# path        path                             type  options
#
/vobs/public  /net/saturn/usr/vobstore/public  mvfs   rw,soft
/vobs/design  /net/saturn/usr/vobstore/design  mvfs   rw,soft
#
# NIS sub-map
#
+current_test_vobs
```

NOTE: The viewroot mount should *not* be specified in an NIS map. This restriction is for reliability: when NIS services are unavailable, you will still want to be able to use views on your host. Be sure to place the viewroot mount entry in your host's standard file system table.

THE CLEARCASE_DOMOUNTS SCRIPT

The *clearcase_domounts* script processes VOB-mount entries, either in a specified ASCII file or in a specified NIS map:

- Use the argument `file` to specify the pathname of an ASCII file
- Use the argument `NIS` to specify the name of an NIS map.

The file or NIS map must be in the format described in section "VOB Mount Entries" above.

The *clearcase_domounts* script uses the "new awk" program, and can use environment variable NAWK to determine the location of this program. If there is no program named *nawk* on your search path, use the EV to specify its full pathname. For example:

```
env NAWK=/usr/bin/awk /usr/atria/etc/clearcase_domounts file /etc/fstab.mvfs
```

EXAMPLES

- Soft-mounted VOB entry in file system table:

```
/usr/src/lib/vob  /vobs/lib   mvfs   rw,soft 0 0
```

- Viewroot mount command with non-default extended naming symbol:

```
mount -t mvfs -o viewroot,xnsuffix=%% /view /view
```

FILES

/etc/fstab
/etc/fstab.mvfs
/etc/init.d/atria

SEE ALSO

cleartool subcommands: mktag, mkview, mkvob, mount, setview, startview, umount
albd_server, init_ccase, exports_ccase, mount(1M), fstab(4)

NAME filesys_osf1 – file system table entries for VOBs: fstab.mvfs (OSF/1)

SYNOPSIS

- VOB mount entry:

  VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

- Non-standard file system type:

  mvfs

- VOB mount options:

  ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **timeo**, **retrans**,
  acregmin, **acregmax**, **acdirmin**, **acdirmax**

- "Obsolete" script for automatically mounting VOBs (*root* only):

  /usr/atria/etc/clearcase_domounts { **file** *filesys-table-pname* | **nis** *NIS-map-name* }

DESCRIPTION

The functionality described in this manual page has been rendered obsolete by the ClearCase VOB registry and the 'cleartool mount' subcommand. Support for this functionality is included in this release, but will be withdrawn in a future release.

To enable access by developers, a *versioned object base* (VOB) must be mounted on a directory as a file system of type *MVFS* (*multiversion file system*). VOB mount entries must be placed in the ClearCase-specific table, */etc/fstab.mvfs*. After a VOB has been mounted on a host, it can be accessed through any *view* that is active on that host.

NOTE: The *fstab.mvfs* file is known only to ClearCase software. It can be processed with the script */usr/atria/etc/clearcase_domounts*, which is described below. The standard UNIX utilities *mount* and *umount* do not process this file at all.

Viewroot Directory

Each host running ClearCase client software must also mount an MVFS file system called the *viewroot directory* (standard name: */view*). The ClearCase startup script performs this task — see the *init_ccase* manual page. There is no way to mount the viewroot directory through an entry in *fstab.mvfs*.

VOB MOUNT ENTRIES

The format of an entry for mounting a VOB closely resembles that for mounting a local file system:

VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

VOB-storage-pathname

Specifies an existing VOB storage directory, (created with *mkvob*). You cannot specify a subdirectory within the VOB; nor can you specify an ancestor directory (such as */usr/src*). This must be a local pathname; a *hostname*:*pathname* specification is invalid. See "Mounting a Remote VOB" below.

VOB-mount-point
Specifies the mount point, which may be any directory. (Typically, it is an empty one.) It must not be the same as the *VOB-storage-pathname* specification.

mvfs The file system type must be **mvfs** (*multiversion file system*).

options All standard UNIX file system table options are supported. See the standard *mount*(1M) manual page for a listing of the default settings; and see "Notes on Mount Options" below.

frequency
pass These parameters have their standard meanings.

Notes on Mount Options

A VOB can be mounted read-write (`rw` option), or read-only (`ro` option).

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` option in the VOB's file system table entry.

Using the `soft` option (recommended) causes ClearCase to return an error if a *view_server* process accessing the VOB goes down and cannot be restarted. Specifying `hard` causes the system to hang in such circumstances.

For VOB mounts, if you don't specify a timeout or retransmission option, a default value is used:

timeo 5 seconds
retrans 7 retries

MOUNTING A REMOTE VOB

A host running ClearCase software can access a VOB that is stored on another ClearCase host in the network. The file system in which the VOB storage directory physically resides must be NFS-mounted on the local host; it does not matter if the NFS mount occurs before or after the associated MVFS mount. Typically, the NFS mount is handled by a file system table entry; however, it could also be auto-mounted.

As with all NFS mounts, the file system containing the VOB storage directory must be listed in its host's */etc/exports* file (see *exports*(4)).

Example:

A VOB storage directory *proj.vbs* is created in the file system */usr/src.export* on remote host *venus*. The following file system table entry on the local host, *saturn*, provides a path to the VOB storage directory:

```
venus:/usr/src.export  /usr/src.import  nfs  rw,soft
```

This entry mounts the VOB as a type-MVFS file system:

```
/usr/src.import/proj.vbs  /usr/src/proj  mvfs rw,soft 0 0
```

CENTRAL ADMINISTRATION OF VOB MOUNT POINTS

An efficient way for a group of hosts in a network to share a set of VOBs is to use a Network Information Services (NIS) map. When it reads the ClearCase-specific file system table *fstab.mvfs*, the *mount_mvfs* utility recognizes a line in the following form as a reference to an NIS map:

+*map-name*

In effect, the entire contents of the map replaces the line. Such NIS map references can occur any number of times in the *fstab.mvfs* file. Moreover, such references can be nested — a map can reference other maps, using lines in the same form.

Example:

```
#
# vob                  mnt                FS      FS
# path                 path               type    options
#
/usr/src/project.vbs   /usr/src/project   mvfs     rw,soft
#
# Use the NIS map "clearcase_vobs" for all other VOB mounts
#
+clearcase_vobs
```

NIS Map Format

Lines in the NIS map have a slightly different format from lines in the file system table. The first two fields are reversed: in the NIS map, the first field specifies the mount point, and the second field specifies the VOB storage area.

The following example shows both the reverse field order and the use of a nested NIS map:

```
#
# mount pt    vob storage                      FS    FS
# path        path                             type  options
#
/vobs/public  /net/saturn/usr/vobstore/public  mvfs   rw,soft
/vobs/design  /net/saturn/usr/vobstore/design  mvfs   rw,soft
#
# NIS sub-map
#
+current_test_vobs
```

NOTE: The viewroot mount should *not* be specified in an NIS map. This restriction is for reliability: when NIS services are unavailable, you will still want to be able to use views on your host. Be sure to place the viewroot mount entry in your host's standard file system table.

THE CLEARCASE_DOMOUNTS SCRIPT

The *clearcase_domounts* script processes VOB-mount entries, either in a specified ASCII file or in a specified NIS map:

- Use the argument `file` to specify the pathname of an ASCII file
- Use the argument `NIS` to specify the name of an NIS map.

The file or NIS map must be in the format described in section "VOB Mount Entries" above.

The *clearcase_domounts* script uses the "new awk" program, and can use environment variable NAWK to determine the location of this program. If there is no program named *gawk* (Gnu awk) on your search path, use the EV to specify its full pathname. For example:

```
env NAWK=/usr/bin/awk /usr/atria/etc/clearcase_domounts file /etc/fstab.mvfs
```

EXAMPLES

- Soft-mounted VOB entry in file system table:

```
/usr/src/lib/vob  /vobs/lib   mvfs   rw,soft 0 0
```

- Viewroot mount command with non-default extended naming symbol:

```
mount -t mvfs -o -o=viewroot,-o=xnsuffix=%% /view /view
```

FILES

/etc/fstab
/etc/fstab.mvfs
/sbin/init.d/atria

SEE ALSO

cleartool subcommands: mktag, mkview, mkvob, mount, setview, startview, umount
albd_server, init_ccase, exports_ccase, mount(1M), fstab(4)

NAME filesys_sun4 – file system table entries for VOBs: fstab.mvfs (SunOS-4)

SYNOPSIS

- VOB mount entry:

  VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

- Non-standard file system type:

  mvfs

- VOB mount options:

  ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **timeo**, **retrans**,
  acregmin, **acregmax**, **acdirmin**, **acdirmax**

- "Obsolete" script for automatically mounting VOBs (*root* only):

  /usr/atria/etc/clearcase_domounts { **file** *filesys-table-pname* | **nis** *NIS-map-name* }

DESCRIPTION

The functionality described in this manual page has been rendered obsolete by the ClearCase VOB registry and the 'cleartool mount' subcommand. Support for this functionality is included in this release, but will be withdrawn in a future release.

To enable access by developers, a *versioned object base* (VOB) must be mounted on a directory as a file system of type *MVFS* (*multiversion file system*). VOB mount entries can be placed in your host's standard file system table, */etc/fstab*, or in the ClearCase-specific table, */etc/fstab.mvfs*. After a VOB has been mounted on a host, it can be accessed through any *view* that is active on that host.

NOTE: The *fstab.mvfs* file is known only to ClearCase software. It can be processed with the script */usr/atria/etc/clearcase_domounts*, which is described below. The standard UNIX utilities *mount* and *umount* do not process this file at all.

Viewroot Directory

Each host running ClearCase client software must also mount an MVFS file system called the *viewroot directory* (standard name: */view*). The ClearCase startup script performs this task — see the *init_ccase* manual page. There is no way to mount the viewroot directory through an entry in *fstab.mvfs*.

VOB MOUNT ENTRIES

The format of an entry for mounting a VOB closely resembles that for mounting a local file system:

VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

VOB-storage-pathname

Specifies an existing VOB storage directory, (created with *mkvob*). You cannot specify a subdirectory within the VOB; nor can you specify an ancestor directory (such as */usr/src*). This must be a local pathname; a *hostname*:*pathname* specification is invalid. See "Mounting a Remote VOB" below.

VOB-mount-point
Specifies the mount point, which may be any directory. (Typically, it is an empty one.) It must not be the same as the *VOB-storage-pathname* specification.

mvfs The file system type must be **mvfs** (*multiversion file system*).

options All standard UNIX file system table options are supported. See the standard *mount*(1M) manual page for a listing of the default settings; and see "Notes on Mount Options" below.

frequency
pass These parameters have their standard meanings.

Notes on Mount Options

A VOB can be mounted read-write (`rw` option), or read-only (`ro` option).

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` option in the VOB's file system table entry.

Using the `soft` option (recommended) causes ClearCase to return an error if a *view_server* process accessing the VOB goes down and cannot be restarted. Specifying `hard` causes the system to hang in such circumstances.

For VOB mounts, if you don't specify a timeout or retransmission option, a default value is used:

timeo 5 seconds
retrans 7 retries

MOUNTING A REMOTE VOB

A host running ClearCase software can access a VOB that is stored on another ClearCase host in the network. The file system in which the VOB storage directory physically resides must be NFS-mounted on the local host; it does not matter if the NFS mount occurs before or after the associated MVFS mount. Typically, the NFS mount is handled by a file system table entry; however, it could also be auto-mounted.

As with all NFS mounts, the file system containing the VOB storage directory must be listed in its host's */etc/exports* file (see *exports*(4)).

Example:

A VOB storage directory *proj.vbs* is created in the file system */usr/src.export* on remote host *venus*. The following file system table entry on the local host, *saturn*, provides a path to the VOB storage directory:

```
venus:/usr/src.export  /usr/src.import  nfs  rw,soft
```

This entry mounts the VOB as a type-MVFS file system:

```
/usr/src.import/proj.vbs  /usr/src/proj  mvfs rw,soft 0 0
```

CENTRAL ADMINISTRATION OF VOB MOUNT POINTS

An efficient way for a group of hosts in a network to share a set of VOBs is to use a Network Information Services (NIS) map. When it reads the ClearCase-specific file system table *fstab.mvfs*, the *mount_mvfs* utility recognizes a line in the following form as a reference to an NIS map:

+*map-name*

In effect, the entire contents of the map replaces the line. Such NIS map references can occur any number of times in the *fstab.mvfs* file. Moreover, such references can be nested — a map can reference other maps, using lines in the same form.

Example:

```
#
# vob                  mnt                FS      FS
# path                 path               type    options
#
/usr/src/project.vbs   /usr/src/project   mvfs     rw,soft
#
# Use the NIS map "clearcase_vobs" for all other VOB mounts
#
+clearcase_vobs
```

NIS Map Format

Lines in the NIS map have a slightly different format from lines in the file system table. The first two fields are reversed: in the NIS map, the first field specifies the mount point, and the second field specifies the VOB storage area.

The following example shows both the reverse field order and the use of a nested NIS map:

```
#
# mount pt    vob storage                      FS    FS
# path        path                             type  options
#
/vobs/public  /net/saturn/usr/vobstore/public  mvfs   rw,soft
/vobs/design  /net/saturn/usr/vobstore/design  mvfs   rw,soft
#
# NIS sub-map
#
+current_test_vobs
```

NOTE: The viewroot mount should *not* be specified in an NIS map. This restriction is for reliability: when NIS services are unavailable, you will still want to be able to use views on your host. Be sure to place the viewroot mount entry in your host's standard file system table.

THE CLEARCASE_DOMOUNTS SCRIPT

The *clearcase_domounts* script processes VOB-mount entries, either in a specified ASCII file or in a specified NIS map:

- Use the argument `file` to specify the pathname of an ASCII file
- Use the argument `NIS` to specify the name of an NIS map.

The file or NIS map must be in the format described in section "VOB Mount Entries" above.

The *clearcase_domounts* script uses the "new awk" program, and can use environment variable NAWK to determine the location of this program. If there is no program named *nawk* on your search path, use the EV to specify its full pathname. For example:

```
env NAWK=/usr/bin/awk /usr/atria/etc/clearcase_domounts file /etc/fstab.mvfs
```

EXAMPLES

- Soft-mounted VOB entry in file system table:

```
/usr/src/lib/vob  /vobs/lib   mvfs   rw,soft 0 0
```

- Viewroot mount command with non-default extended naming symbol:

```
mount -t mvfs -o viewroot,xnsuffix=%% /view /view
```

FILES

/etc/fstab
/etc/fstab.mvfs
/etc/rc.atria

SEE ALSO

cleartool subcommands: mktag, mkview, mkvob, mount, setview, startview, umount
albd_server, init_ccase, exports_ccase, mount(1M), fstab(5)

NAME filesys_sun5 – file system table entries for VOBs: fstab.mvfs (SunOS-5)

SYNOPSIS

- VOB mount entry:

  VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

- Non-standard file system type:

  mvfs

- VOB mount options:

  ro, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **timeo**, **retrans**,
  acregmin, **acregmax**, **acdirmin**, **acdirmax**

- "Obsolete" script for automatically mounting VOBs (*root* only):

  /usr/atria/etc/clearcase_domounts { **file** *filesys-table-pname* | **nis** *NIS-map-name* }

DESCRIPTION

The functionality described in this manual page has been rendered obsolete by the ClearCase VOB registry and the 'cleartool mount' subcommand. Support for this functionality is included in this release, but will be withdrawn in a future release.

To enable access by developers, a *versioned object base* (VOB) must be mounted on a directory as a file system of type *MVFS* (*multiversion file system*). VOB mount entries can be placed in your host's standard file system table, */etc/vfstab*, or in the ClearCase-specific table, */etc/fstab.mvfs*. After a VOB has been mounted on a host, it can be accessed through any *view* that is active on that host.

NOTE: The *fstab.mvfs* file is known only to ClearCase software. It can be processed with the script */usr/atria/etc/clearcase_domounts*, which is described below. The standard UNIX utilities *mount*, *umount*, and *mountall* do not process this file at all.

Viewroot Directory

Each host running ClearCase client software must also mount an MVFS file system called the *viewroot directory* (standard name: */view*). The ClearCase startup script performs this task — see the *init_ccase* manual page. There is no way to mount the viewroot directory through an entry in *fstab.mvfs*.

VOB MOUNT ENTRIES

The format of an entry for mounting a VOB closely resembles that for mounting a local file system:

VOB-storage-pathname *VOB-mount-point* **mvfs** *options* *frequency* *pass*

VOB-storage-pathname

Specifies an existing VOB storage directory, (created with *mkvob*). You cannot specify a subdirectory within the VOB; nor can you specify an ancestor directory (such as */usr/src*). This must be a local pathname; a *hostname*:*pathname* specification is invalid. See "Mounting a Remote VOB" below.

VOB-mount-point
Specifies the mount point, which may be any directory. (Typically, it is an empty one.) It must not be the same as the *VOB-storage-pathname* specification.

mvfs The file system type must be **mvfs** (*multiversion file system*).

options All standard UNIX file system table options are supported. See the standard *mount*(1M) manual page for a listing of the default settings; and see "Notes on Mount Options" below.

frequency
pass These parameters have their standard meanings.

Notes on Mount Options

A VOB can be mounted read-write (`rw` option), or read-only (`ro` option).

By default, a VOB is mounted in `nointr` mode. This means that operations on MVFS files (for example, *open*(2)) cannot be interrupted by typing the INTR character (typically, `<Ctrl-C>`). To enable keyboard interrupts of such operations, use the `intr` option in the VOB's file system table entry.

Using the `soft` option (recommended) causes ClearCase to return an error if a *view_server* process accessing the VOB goes down and cannot be restarted. Specifying `hard` causes the system to hang in such circumstances.

For VOB mounts, if you don't specify a timeout or retransmission option, a default value is used:

timeo 5 seconds
retrans 7 retries

MOUNTING A REMOTE VOB

A host running ClearCase software can access a VOB that is stored on another ClearCase host in the network. The file system in which the VOB storage directory physically resides must be NFS-mounted on the local host; it does not matter if the NFS mount occurs before or after the associated MVFS mount. Typically, the NFS mount is handled by a file system table entry; however, it could also be auto-mounted.

As with all NFS mounts, the file system containing the VOB storage directory must be listed in its host's */etc/dfs/dfstab* file (see *exports*(4)).

Example:

A VOB storage directory *proj.vbs* is created in the file system */usr/src.export* on remote host *venus*. The following file system table entry on the local host, *saturn*, provides a path to the VOB storage directory:

```
venus:/usr/src.export  /usr/src.import  nfs  rw,soft
```

This entry mounts the VOB as a type-MVFS file system:

```
/usr/src.import/proj.vbs  /usr/src/proj  mvfs rw,soft 0 0
```

CENTRAL ADMINISTRATION OF VOB MOUNT POINTS

An efficient way for a group of hosts in a network to share a set of VOBs is to use a Network Information Services (NIS) map. When it reads the ClearCase-specific file system table *fstab.mvfs*, the *mount_mvfs* utility recognizes a line in the following form as a reference to an NIS map:

+*map-name*

In effect, the entire contents of the map replaces the line. Such NIS map references can occur any number of times in the *fstab.mvfs* file. Moreover, such references can be nested — a map can reference other maps, using lines in the same form.

Example:

```
#
# vob                  mnt                FS      FS
# path                 path               type    options
#
/usr/src/project.vbs   /usr/src/project   mvfs     rw,soft
#
# Use the NIS map "clearcase_vobs" for all other VOB mounts
#
+clearcase_vobs
```

NIS Map Format

Lines in the NIS map have a slightly different format from lines in the file system table. The first two fields are reversed: in the NIS map, the first field specifies the mount point, and the second field specifies the VOB storage area.

The following example shows both the reverse field order and the use of a nested NIS map:

```
#
# mount pt    vob storage                      FS    FS
# path        path                             type  options
#
/vobs/public  /net/saturn/usr/vobstore/public  mvfs   rw,soft
/vobs/design  /net/saturn/usr/vobstore/design  mvfs   rw,soft
#
# NIS sub-map
#
+current_test_vobs
```

NOTE: The viewroot mount should *not* be specified in an NIS map. This restriction is for reliability: when NIS services are unavailable, you will still want to be able to use views on your host. Be sure to place the viewroot mount entry in your host's standard file system table.

THE CLEARCASE_DOMOUNTS SCRIPT

The *clearcase_domounts* script processes VOB-mount entries, either in a specified ASCII file or in a specified NIS map:

- Use the argument `file` to specify the pathname of an ASCII file
- Use the argument `NIS` to specify the name of an NIS map.

The file or NIS map must be in the format described in section "VOB Mount Entries" above.

The *clearcase_domounts* script uses the "new awk" program, and can use environment variable NAWK to determine the location of this program. If there is no program named *nawk* on your search path, use the EV to specify its full pathname. For example:

```
env NAWK=/usr/bin/awk /usr/atria/etc/clearcase_domounts file /etc/fstab.mvfs
```

EXAMPLES

- Soft-mounted VOB entry in file system table:

```
/usr/src/lib/vob  /vobs/lib   mvfs   rw,soft 0 0
```

- Viewroot mount command with non-default extended naming symbol:

```
mount -t mvfs -o viewroot,xnsuffix=%% /view /view
```

FILES

/etc/vfstab
/etc/fstab.mvfs
/etc/init.d/atria

SEE ALSO

cleartool subcommands: mktag, mkview, mkvob, mount, setview, startview, umount
albd_server, init_ccase, exports_ccase, mount(1M), vfstab(4)

NAME fmt_ccase – format strings for cleartool command output

SYNOPSIS

- –fmt option syntax:

  cleartool *subcommand* **–fmt** *format-string other-subcommand-options-and-args*

- *subcommand* is one of various reporting commands (*annotate*, *describe*, *lshistory*, *lscheckout*, and so on)

- *format-string* is a quoted character string, composed of alphanumeric characters, *conversion specifications*, and *escape sequences*.

  Conversion specifications:

  %a attributes (modifiers: N, S, [*attr-type*])
  %c comment string (modifiers: N)
  %d date (modifiers: S, V, DA, MA, BA, OA)
  %e event description
  %h host name
  %l labels (modifiers: C, N)
  %n name of object (modifiers: E, L, S, PS, V, PV, X)
  %m object kind (version, derived object, and so on)
  %o operation kind (checkin, lock, mkelem, and so on)
  %u user/group information (modifiers: F, G, L)
  %% % character

  Escape sequences:

  \n <NL>
  \t <Tab>
  \' single quote
  \\ literal backslash
  nnn character specified by octal code

DESCRIPTION

Many *cleartool* subcommands read information from a VOB database, format the data, and send it to standard output. (In most cases, the information is stored in *event records*, written by *cleartool* when it creates or modifies an object in a VOB. See the *events_ccase* manual page.) Some of these subcommands have a –fmt option, which you can use to format simple reports on VOB contents. Note that –fmt is a mutually exclusive alternative to the –short and –long options.

The following example shows how output-formatting options affect an *lshistory* command.

```
% cleartool lshistory –since 1–Feb util.c
10-Feb.11:21   anne      create version "util.c@@/main/rel2_bugfix/1"
  "fix bug: extra NL in time string"
10-Feb.11:21   anne      create version "util.c@@/main/rel2_bugfix/0"
10-Feb.11:21   anne      create branch "util.c@@/main/rel2_bugfix"
```

```
% cleartool lshistory –short –since 1–Feb util.c
util.c@@/main/rel2_bugfix/1
util.c@@/main/rel2_bugfix/0
util.c@@/main/rel2_bugfix

% cleartool lshistory –fmt "\tElement: %–13.13En  Version: %Vn\n" –since 1–Feb util.c
        Element: util.c         Version: /main/rel2_bugfix/1
        Element: util.c         Version: /main/rel2_bugfix/0
        Element: util.c         Version: /main/rel2_bugfix
```

(A `\t` escape sequence tabs output to the next tab stop. Tab stops occur at eight-character intervals, except as described in the *annotate* manual page.)

The 'describe' Command and -fmt

The *describe* subcommand has its own output-formatting options: `–predecessor`, `–alabel`, `–aattr`, and `–ahlink`. With one or more of these options, *describe* replaces its standard output format with:

- A "header line", which lists the object's name.
- The predecessor version-ID, version labels, attribute values, and/or names of hyperlinked objects, as specified by the options.

If you combine `–fmt` with any of these options, *describe* uses the *format-string* to construct and display the header line.

CONVERSION SPECIFICATIONS

A *conversion specification* identifies a particular data item to display and specifies its display format.

Syntax

%[*min*][*.max*][*MODIFIER* [, …]]*keyletter*

The conversion specification format closely resembles that of the C-language function *printf*(3):

- percent character (%)
- optionally, a minimum and/or maximum field display width specifier, of the form *min.max* (see "Specifying Field Width" below)
- optionally (for some conversion specs), one or more modifier characters (uppercase) that specify one or more variants, *and/or*, a bracket-enclosed parameter (see %a)
- a keyletter (lowercase), which indicates the kind of data to display

Unlike *printf*(3) specifiers, conversion specifications are not replaced by arguments supplied elsewhere on the command line; they are replaced automatically by *cleartool*, usually with field values extracted from event records.

The conversion specifications are:

%n **Name of object** — for a file system object, the extended pathname (including the version-ID for versions, and the DO-ID for derived objects); for a type object, its name. Variants:

- **%En** **Element name** — for a file system object, its standard file or element name, or its pathname; for a type object, its name.
- **%Ln** **Leaf name** — for any named object, its simple name. The terminal node of a pathname. This modifier can be combined with others.
- **%Sn** **Short name** — for a version, a short form of the version-ID: *branch/version*. The null string otherwise.
- **%PSn** **Predecessor Short name** — for a version, a short form of the predecessor version's version-ID: *branch/version*. The null string otherwise.
- **%Vn** **Version ID** — for a version or derived object, the version-ID; the null string otherwise.
- **%PVn** **Predecessor Version ID** — for a version, the predecessor version's version-ID; the null string otherwise.
- **%Xn** **Extended name** — same as default **%n** output, but for checked-out versions, append the extension `@@`*/branch*`/CHECKEDOUT`.

%a **Attributes** — for elements, branches, or versions, all attached attributes; the null string otherwise. Attributes are listed as *attr=value* pairs. These pairs are enclosed in parentheses and separated by a comma-space combination (`,<SP>`). Variants:

- **%Na** **No commas** — suppress the parentheses and commas in attribute list output; separate multiple attributes with spaces only.
- **%Sa** **Value only** — display attribute values only (rather than *attr=value* pairs).
- **%[***attype***]a** **This attribute only** — display only the specified attribute, if it has been attached to the object.

%c **Comment string** — the user-supplied or system-generated comment stored in an event record. A newline character is appended to the comment string for display purposes only. Variant:

- **%Nc** **No newline** — do not append a newline character to the comment string.

%d **Date/Time** — the timestamp of the operation or event, in *date*.*time* format. Variants:

%Sd (short) Date only.

%Vd (very long) Day of week, date, and time.

%DAd Age in days.

%MAd Age in months.

%BAd Age as a bar graph (longer bars for more recent events). A bar graph is drawn as a sequence of 0-5 # characters, representing the elapsed time since the reported operation as follows:

#####	less than a week
####	less than a month
###	less than a three months
##	less than six months
#	less than a year
	more than a year

%OAd Age as a bar graph (longer bars for older events). A bar graph is drawn as a sequence of 0-5 # characters, representing the elapsed time since the reported operation as follows:

#####	more than a year
####	less than a year
###	less than six months
##	less than three months
#	less than a month
	less than a week

%h **Host name** — as reported by `uname -n`.

%l **Labels** — for versions, all attached labels; the null string otherwise. Labels are output as a comma-separated list, enclosed in parentheses. A <Space> character follows each comma. Variants:

%Cl **Max labels** — specify the maximum number of labels to display with the *max-field-width* parameter (see "Specifying Field Width" below). If there are more labels, "..." is appended to the output. If no *max-field-width* is specified, the maximum defaults to three.

%Nl **No commas** — suppress the parentheses and commas in label list output; separate labels with spaces only.

%m **Object kind** — the kind of object involved in the operation, for example:

```
file element
branch
version
derived object
branch type
label type
```

%o **Operation kind** — the ClearCase operation that caused the event to take place; commonly, the name of a *cleartool* subcommand. For example:

```
mkelem
mklabel
checkin
checkout
```

See the *events_ccase* manual page for a complete list of operations and the commands that cause them.

%e **Event kind** — a brief description of the event. The event kind is derived programmatically from an event record's name, object kind, and operation kind fields. Sample event kinds:

```
create version
create branch
make hyperlink "Merge" on version
make label "REL2" on version
lock branch type
```

%u **User information** — the login name of the user associated with the event or object. Variants:

%Fu **Full name** — the user's full name, extracted from the password database.

%Gu **Group name** — only the user's group name.

%Lu **Long name** — the user login name and group (*user.group*).

%% Percent character (%).

Specifying Field Width

A conversion specification can include an optional *field width specifier*, which assigns a minimum and/or maximum width, in characters, to the data field display. For example, the conversion specifier `%10.15Lu` will display, for each output line, the user's login name and group with a minimum of 10 characters (space padded if necessary) but not more than 15.

Usage rules:

- A single number is interpreted as a minimum width.
- To supply only a maximum width, precede the number with a decimal point (for example, `%.10En`) or with a zero and decimal point (`%0.10En`).
- To specify a constant display width, set the minimum and maximum widths to the same value (`%20.20c`).
- Values smaller than the specified minimum width are right justified (padded left). A negative minimum width value (`%-20.20c`) left justifies short values.
- Values longer than the specified maximum width are truncated from the right. A negative maximum width value (`%15.-15Sn`) truncates long values from the left.
- A maximum width specifier has special meaning when used with the **%Cl** specifier. For example, `%.5Cl` prints a version's first five labels only, followed by "...".

EXAMPLES

- Format the output from `lsco -cview`.

```
% cleartool lsco –cview –fmt "\t%–10.10n (from %8.8PVn)  %d   %u\n"
        util.c    (from /main/23)  24-Jun-94.14:12:48   anne
        main.c    (from /main/46)  23-Jun-94.18:42:33   anne
        msg.c     (from ugfix/11)  23-Jun-94.10:45:13   anne
        msg.h     (from bugfix/3)  22-Jun-94.14:51:55   anne
```

- Format the event history of a file element. (The command line, including the quoted format sting, constitutes a single input line. The input line below is broken to improve readability. Spaces are significant.)

```
% cleartool lshistory –fmt "OBJ–NAME: %–20.20n\n  USER: %–8.8u\n  DATE: %d\n
     OPERATION:\t%–12.12o\n     OBJ–TYPE:\t%–15.15m\n     EVENT:\t%e\n
     COMMENT: %c\n" util.c
OBJ-NAME: util.c@@/main/3
  USER: anne
  DATE: 10-May-94.09:24:38
    OPERATION:  checkin
    OBJ-TYPE:   version
    EVENT:      create version
  COMMENT: fix bug r2-307

OBJ-NAME: util.c@@/main/2
  USER: anne
  DATE: 10-May-94.09:09:29
    OPERATION:  checkin
    OBJ-TYPE:   version
    EVENT:      create version
  COMMENT: ready for code review
...
```

- Mimic the output from `lshistory -long`. Note the use of single quotes to enclose the format string, which includes literal double quotes.

```
% cleartool lshistory –fmt '%d     %Fu (%u@%h)\n  %e "%n"\n  "%Nc"\n' util.c
11-May-94.09:24:38     Anne Duvo (anne@neptune)
  create version "util.c@@/main/3"
  "fix bug r2-307"
10-May-94.09:09:29     Ravi Singha (ravi@mercury)
  create version "util.c@@/main/2"
  "ready for code review"
...
```

- Describe the element *main.c* in detail. This example illustrates the complete set of conversion specifiers (but does not use field width specifiers). Again, the command is a single input line; line breaks are added for readability.

```
% cleartool describe –fmt "Name (default): %n\n
     Element name: %En\n
     Leaf name: %Ln\n
     Short name: %Sn\n
     Predecessor short name: %PSn\n
     Version ID: %Vn\n
     Predecessor version ID: %PVn\n
     Extended name: %Xn\n
     Attributes: %a\n
     Attr values only: %Sa\n
```

```
    Attrs without commas or parens: %Na\n
    This attr only: %[Tested]a\n
    Comment: %c
    Date/Time:\tdefault: %d\n
    \t\tshort: %Sd\n
    \t\tlong: %Vd\n
    Age in days: %DAd\n
    Age in months: %MAd\n
    Age graph (long = new): %BAd\n
    Age graph (long = old): %OAd\n
    Host: %h\n
    Labels: %Cl\n
    Labels without commas or parens: %Nl\n
    Object kind: %m\n
    Operation kind: %o\n
    Event kind: %e\n
    User (default): %u\n
    Full user name: %Fu\n
    Group name: %Gu\n
    Long name: %Lu\n\n"   main.c
Name (default): main.c@@/main/34
Element name: main.c
Leaf name: 34
Short name: /main/34
Predecessor short name: /main/33
Version ID: /main/34
Predecessor version ID: /main/33
Extended name: main.c@@/main/34
Attributes: (Tested="yes", QAlevel=4, Responsible="anne")
Attr values only: ("yes", 4, "anne")
Attrs without commas or parens: Tested="yes" QAlevel=4 Responsible="anne"
This attr only: (Tested="yes")
Comment: still needs QA
Date/Time:      default: 30-Jul-94.15:02:49
                short: 30-Jul-94
                long: Tuesday 07/30/94 15:02:49
Age in days: 42
Age in months: 1
Age graph (long = new): ####
Age graph (long = old): ##
Host: neptune
Labels: (Rel3.1C, Rel3.1D, Rel3.1E)
Labels without commas or parens: Rel3.1C Rel3.1D Rel3.1E
Object kind: version
Operation kind: checkin
Event kind: create version
User (default): anne
Full user name: Anne Duvo
Group name: dev
Long name: anne.dev
```

SEE ALSO

cleartool subcommands: lshistory, describe, lscheckout, lsdo, annotate, lslock, lspool, lsreplica, lstype
events_ccase, printf(3)

NAME init_ccase – ClearCase startup/shutdown script

DESCRIPTION

The scripts that start ClearCase processes at system startup, and kill those processes at system shutdown, are architecture-specific. Consult the following manual pages:

SunOS-4	init_sun4
SunOS-5	init_sun5
HPUX-9	init_hpx9
IRIX-5	init_irx5
OSF/1 V2	init_osf1

NAME init_hpx9 – ClearCase startup/shutdown script (HPUX-9)

SYNOPSIS

/etc/rc.atria { **start** | **stop** }

DESCRIPTION

The shell script */etc/rc.atria* is invoked automatically at system startup and shutdown. It can also be executed as a shell command.

CLEARCASE STARTUP

When invoked with the argument `start` (or without an argument), the script performs ClearCase initialization:

- start the Location Broker Daemon, *albd_server*
- start the database lock manager process, *lockmgr*
- initialize the viewroot directory (default name */view*)
- mount public VOBs listed in ClearCase storage registry. If the network is partitioned into multiple *network regions*, only the VOBs that have public VOB-tags in the local host's region will be mounted.
- export VOBs through particular views to enable access by non-ClearCase hosts; the list of VOBs to be exported is read from the ClearCase-specific file, */etc/exports.mvfs*.

Startup Retry Loop

The startup script itself resides outside the host's ClearCase installation area (by default, */usr/atria*). The actual work, however, is performed by a script that resides *inside* the installation area: */usr/atria/etc/rc.atria*. If this script cannot be accessed (for example, because it is actually located on a remote host that is currently down), the startup script enters a retry loop: it periodically attempts to start ClearCase processing over an extended period (about half an hour). If the final retry fails, an error message is displayed.

You can break the startup script out of its retry loop by removing the flag file */tmp/ClearCase.retrying*.

The Viewroot Mount Command

The startup script runs a standard *mount* command to mount the *viewroot* directory as a file system of type MVFS:

```
mount -t mvfs -o rw,viewroot /view /view
```

You can change the extending naming symbol by appending a string to the argument that follows the `–o` option:

,xnsuffix=*symbol*

This specifies a character string to be used on the local host as the ClearCase *extended naming symbol*. By default, the string `@@` is used. Be careful: this option affects the local host only; other hosts may use the default extended naming symbol or another symbol specified with this mount option.

You can specify a directory other than */view* as the viewroot. Whatever directory you specify (for example, */ccasevu*) must exist at system startup time. Note that you must specify this directory name twice in the *mount* command.

Mounting the viewroot directory enables use of ClearCase views on the local host. When a view is activated (by *startview*, *setview*, or *mktag*), its view-tag is entered into the viewroot directory. For example, activating a view whose view–tag is *gamma* would create directory entry */view/gamma*. See the *pathnames_ccase* manual page for a discussion of *view-extended pathnames* that use such directory entries.

A mounted viewroot directory is not actually an on-disk directory. Rather, it is a data structure maintained in main memory by the MVFS code linked with the operating system kernel. The viewroot directory's list of view-tags is lost whenever ClearCase operation on the local host is stopped (including an operating system shutdown).

The viewroot directory cannot be exported, and cannot be mounted by any other host. Each ClearCase host must have its own viewroot directory.

CLEARCASE SHUTDOWN

When invoked with the argument `stop`, the script performs ClearCase shutdown:

- unmount all VOBs
- kill the *vob_server* processes for VOB's whose storage directories are on the local host
- kill the *albd_server* process, which also causes *view_server*, *db_server*, and *vobrpc_server* processes to exit
- kill the *lockmgr* process
- unmount the viewroot directory

SEE ALSO

albd_server, exports_ccase, filesys_ccase, lockmgr, mount_ccase, pathnames_ccase

NAME init_irx5 – ClearCase startup/shutdown script (IRIX-5)

SYNOPSIS

/etc/init.d/atria { **start** | **stop** }

DESCRIPTION

The shell script */etc/init.d/atria* is invoked automatically at system startup and shutdown. It can also be executed as a shell command.

CLEARCASE STARTUP

When invoked with the argument `start` (or without an argument), the script performs ClearCase initialization:

- start the Location Broker Daemon, *albd_server*
- start the database lock manager process, *lockmgr*
- initialize the viewroot directory (default name */view*)
- mount public VOBs listed in ClearCase storage registry. If the network is partitioned into multiple *network regions*, only the VOBs that have public VOB-tags in the local host's region will be mounted.
- export VOBs through particular views to enable access by non-ClearCase hosts; the list of VOBs to be exported is read from the ClearCase-specific file, */etc/exports.mvfs*.

Startup Retry Loop

The startup script itself resides outside the host's ClearCase installation area (by default, */usr/atria*). The actual work, however, is performed by a script that resides *inside* the installation area: */usr/atria/etc/init.d/atria*. If this script cannot be accessed (for example, because it is actually located on a remote host that is currently down), the startup script enters a retry loop: it periodically attempts to start ClearCase processing over an extended period (about half an hour). If the final retry fails, an error message is displayed.

You can break the startup script out of its retry loop by removing the flag file */tmp/ClearCase.retrying*.

The Viewroot Mount Command

The startup script runs a standard *mount* command to mount the *viewroot* directory as a file system of type MVFS:

```
mount -t mvfs -o rw,viewroot /view /view
```

You can change the extending naming symbol by appending a string to the argument that follows the `–o` option:

,xnsuffix=*symbol*

This specifies a character string to be used on the local host as the ClearCase *extended naming symbol*. By default, the string `@@` is used. Be careful: this option affects the local host only; other hosts may use the default extended naming symbol or another symbol specified with this mount option.

You can specify a directory other than */view* as the viewroot. Whatever directory you specify (for example, */ccasevu*) must exist at system startup time. Note that you must specify this directory name twice in the *mount* command.

Mounting the viewroot directory enables use of ClearCase views on the local host. When a view is activated (by *startview*, *setview*, or *mktag*), its view-tag is entered into the viewroot directory. For example, activating a view whose view–tag is *gamma* would create directory entry */view/gamma*. See the *pathnames_ccase* manual page for a discussion of *view-extended pathnames* that use such directory entries.

A mounted viewroot directory is not actually an on-disk directory. Rather, it is a data structure maintained in main memory by the MVFS code linked with the operating system kernel. The viewroot directory's list of view-tags is lost whenever ClearCase operation on the local host is stopped (including an operating system shutdown).

The viewroot directory cannot be exported, and cannot be mounted by any other host. Each ClearCase host must have its own viewroot directory.

CLEARCASE SHUTDOWN

When invoked with the argument `stop`, the script performs ClearCase shutdown:

- unmount all VOBs
- kill the *vob_server* processes for VOB's whose storage directories are on the local host
- kill the *albd_server* process, which also causes *view_server*, *db_server*, and *vobrpc_server* processes to exit
- kill the *lockmgr* process
- unmount the viewroot directory

SEE ALSO

albd_server, exports_ccase, filesys_ccase, lockmgr, mount_ccase, pathnames_ccase

NAME init_osf1 – ClearCase startup/shutdown script (OSF/1)

SYNOPSIS

/sbin/init.d/atria { **start** | **stop** }

DESCRIPTION

The shell script */sbin/init.d/atria* is invoked automatically at system startup and shutdown. It can also be executed as a shell command.

CLEARCASE STARTUP

When invoked with the argument `start` (or without an argument), the script performs ClearCase initialization:

- start the Location Broker Daemon, *albd_server*
- start the database lock manager process, *lockmgr*
- initialize the viewroot directory (default name */view*)
- mount public VOBs listed in ClearCase storage registry. If the network is partitioned into multiple *network regions*, only the VOBs that have public VOB-tags in the local host's region will be mounted.
- export VOBs through particular views to enable access by non-ClearCase hosts; the list of VOBs to be exported is read from the standard exports file, */etc/exports*.

Startup Retry Loop

The startup script itself resides outside the host's ClearCase installation area (by default, */usr/atria*). The actual work, however, is performed by a script that resides *inside* the installation area: */usr/atria/sbin/init.d/atria*. If this script cannot be accessed (for example, because it is actually located on a remote host that is currently down), the startup script enters a retry loop: it periodically attempts to start ClearCase processing over an extended period (about half an hour). If the final retry fails, an error message is displayed.

You can break the startup script out of its retry loop by removing the flag file */tmp/ClearCase.retrying*.

The Viewroot Mount Command

The startup script runs a standard *mount* command to mount the *viewroot* directory as a file system of type MVFS:

```
mount -t mvfs -o -o=rw,-o=viewroot /view /view
```

You can change the extending naming symbol by appending a string to the argument that follows the `–o` option:

,-o=xnsuffix=*symbol*

This specifies a character string to be used on the local host as the ClearCase *extended naming symbol*. By default, the string `@@` is used. Be careful: this option affects the local host only; other hosts may use the default extended naming symbol or another symbol specified with this mount option.

Mounting the viewroot directory enables use of ClearCase views on the local host. When a view is activated (by *startview*, *setview*, or *mktag*), its view-tag is entered into the viewroot directory. For example, activating a view whose view–tag is *gamma* would create directory entry */view/gamma*. See the *pathnames_ccase* manual page for a discussion of *view-extended pathnames* that use such directory entries.

A mounted viewroot directory is not actually an on-disk directory. Rather, it is a data structure maintained in main memory by the MVFS code linked with the operating system kernel. The viewroot directory's list of view-tags is lost whenever ClearCase operation on the local host is stopped (including an operating system shutdown).

The viewroot directory cannot be exported, and cannot be mounted by any other host. Each ClearCase host must have its own viewroot directory.

CLEARCASE SHUTDOWN

When invoked with the argument `stop`, the script performs ClearCase shutdown:

- unmount all VOBs
- kill the *vob_server* processes for VOB's whose storage directories are on the local host
- kill the *albd_server* process, which also causes *view_server*, *db_server*, and *vobrpc_server* processes to exit
- kill the *lockmgr* process
- kill all user processes that are using the MVFS (multiversion file system)
- unmount the viewroot directory

SEE ALSO

albd_server, exports_ccase, filesys_ccase, lockmgr, mount_ccase, pathnames_ccase

NAME init_sun4 – ClearCase startup/shutdown script (SunOS-4)

SYNOPSIS

/etc/rc.atria { **start** | **stop** }

DESCRIPTION

The shell script */etc/rc.atria* is invoked automatically at system startup and shutdown. It can also be executed as a shell command.

CLEARCASE STARTUP

When invoked with the argument `start` (or without an argument), the script performs ClearCase initialization:

- dynamically load the MVFS (multiversion file system) into the operating system kernel
- start the Location Broker Daemon, *albd_server*
- start the database lock manager process, *lockmgr*
- initialize the viewroot directory (default name */view*)
- mount public VOBs listed in ClearCase storage registry. If the network is partitioned into multiple *network regions*, only the VOBs that have public VOB-tags in the local host's region will be mounted.
- export VOBs through particular views to enable access by non-ClearCase hosts; the list of VOBs to be exported is read from the ClearCase-specific file, */etc/exports.mvfs*.

Startup Retry Loop

The startup script itself resides outside the host's ClearCase installation area (by default, */usr/atria*). The actual work, however, is performed by a script that resides *inside* the installation area: */usr/atria/etc/rc.atria*. If this script cannot be accessed (for example, because it is actually located on a remote host that is currently down), the startup script enters a retry loop: it periodically attempts to start ClearCase processing over an extended period (about half an hour). If the final retry fails, an error message is displayed.

You can break the startup script out of its retry loop by removing the flag file */tmp/ClearCase.retrying*.

The Viewroot Mount Command

The startup script runs a standard *mount* command to mount the *viewroot* directory as a file system of type MVFS:

```
mount -t mvfs -o rw,viewroot /view /view
```

You can change the extending naming symbol by appending a string to the argument that follows the `–o` option:

,xnsuffix=*symbol*

This specifies a character string to be used on the local host as the ClearCase *extended naming symbol*. By default, the string `@@` is used. Be careful: this option affects the local host only; other hosts may use the default extended naming symbol or another symbol specified with this mount option.

You can specify a directory other than */view* as the viewroot. Whatever directory you specify (for example, */ccasevu*) must exist at system startup time. Note that you must specify this directory name twice in the *mount* command.

Mounting the viewroot directory enables use of ClearCase views on the local host. When a view is activated (by *startview*, *setview*, or *mktag*), its view-tag is entered into the viewroot directory. For example, activating a view whose view–tag is *gamma* would create directory entry */view/gamma*. See the *pathnames_ccase* manual page for a discussion of *view-extended pathnames* that use such directory entries.

A mounted viewroot directory is not actually an on-disk directory. Rather, it is a data structure maintained in main memory by the MVFS code linked with the operating system kernel. The viewroot directory's list of view-tags is lost whenever ClearCase operation on the local host is stopped (including an operating system shutdown).

The viewroot directory cannot be exported, and cannot be mounted by any other host. Each ClearCase host must have its own viewroot directory.

CLEARCASE SHUTDOWN

When invoked with the argument `stop`, the script performs ClearCase shutdown:

- unmount all VOBs
- kill the *vob_server* processes for VOB's whose storage directories are on the local host
- kill the *albd_server* process, which also causes *view_server*, *db_server*, and *vobrpc_server* processes to exit
- kill the *lockmgr* process
- kill all user processes that are using the MVFS (multiversion file system)
- unload the MVFS from the operating system kernel
- unmount the viewroot directory

SEE ALSO

albd_server, exports_ccase, filesys_ccase, lockmgr, mount_ccase, pathnames_ccase

NAME init_sun5 – ClearCase startup/shutdown script (SunOS-5)

SYNOPSIS

/etc/init.d/atria { **start** | **stop** }

DESCRIPTION

The shell script */etc/init.d/atria* is invoked automatically at system startup and shutdown. It can also be executed as a shell command.

CLEARCASE STARTUP

When invoked with the argument `start` (or without an argument), the script performs ClearCase initialization:

- dynamically load the MVFS (multiversion file system) into the operating system kernel
- start the Location Broker Daemon, *albd_server*
- start the database lock manager process, *lockmgr*
- initialize the viewroot directory (default name */view*)
- mount public VOBs listed in ClearCase storage registry. If the network is partitioned into multiple *network regions*, only the VOBs that have public VOB-tags in the local host's region will be mounted.
- export VOBs through particular views to enable access by non-ClearCase hosts; the list of VOBs to be exported is read from the ClearCase-specific file, */etc/exports.mvfs*.

Startup Retry Loop

The startup script itself resides outside the host's ClearCase installation area (by default, */usr/atria*). The actual work, however, is performed by a script that resides *inside* the installation area: */usr/atria/etc/init.d/atria*. If this script cannot be accessed (for example, because it is actually located on a remote host that is currently down), the startup script enters a retry loop: it periodically attempts to start ClearCase processing over an extended period (about half an hour). If the final retry fails, an error message is displayed.

You can break the startup script out of its retry loop by removing the flag file */tmp/ClearCase.retrying*.

The Viewroot Mount Command

The startup script runs a standard *mount* command to mount the *viewroot* directory as a file system of type MVFS:

```
mount -t mvfs -o rw,viewroot /view /view
```

You can change the extending naming symbol by appending a string to the argument that follows the `–o` option:

,xnsuffix=*symbol*

This specifies a character string to be used on the local host as the ClearCase *extended naming symbol*. By default, the string `@@` is used. Be careful: this option affects the local host only; other hosts may use the default extended naming symbol or another symbol specified with this mount option.

You can specify a directory other than */view* as the viewroot. Whatever directory you specify (for example, */ccasevu*) must exist at system startup time. Note that you must specify this directory name twice in the *mount* command.

Mounting the viewroot directory enables use of ClearCase views on the local host. When a view is activated (by *startview*, *setview*, or *mktag*), its view-tag is entered into the viewroot directory. For example, activating a view whose view–tag is *gamma* would create directory entry */view/gamma*. See the *pathnames_ccase* manual page for a discussion of *view-extended pathnames* that use such directory entries.

A mounted viewroot directory is not actually an on-disk directory. Rather, it is a data structure maintained in main memory by the MVFS code linked with the operating system kernel. The viewroot directory's list of view-tags is lost whenever ClearCase operation on the local host is stopped (including an operating system shutdown).

The viewroot directory cannot be exported, and cannot be mounted by any other host. Each ClearCase host must have its own viewroot directory.

CLEARCASE SHUTDOWN

When invoked with the argument `stop`, the script performs ClearCase shutdown:

- unmount all VOBs
- kill the *vob_server* processes for VOB's whose storage directories are on the local host
- kill the *albd_server* process, which also causes *view_server*, *db_server*, and *vobrpc_server* processes to exit
- kill the *lockmgr* process
- kill all user processes that are using the MVFS (multiversion file system)
- unload the MVFS from the operating system kernel
- unmount the viewroot directory

SEE ALSO

albd_server, exports_ccase, filesys_ccase, lockmgr, mount_ccase, pathnames_ccase

NAME license.db – ClearCase network-wide license database

SYNOPSIS

- Specify a set of licenses:

  -license ClearCase *vendor* **any.***max-users expiration-date password*

- Specify license timeout period:

  –timeout *minutes*

- Specify users' license priorities:

  -user { *user-name* | *user-ID* } …

- Forbid ClearCase use by certain users:

  –nuser *user-name* …

- Enable auditing of licensing activity:

  -audit

DESCRIPTION

NOTE: Some ClearCase hosts do not use the Atria-provided licensing scheme.

One or more hosts in the network must be designated as ClearCase *license server hosts*. Each one must also be an *installation host*: a host on which the ClearCase software is installed. Network-wide licensing of ClearCase usage is established as follows:

- **On a license server host** — creating (or appending data to) a text file named */usr/adm/atria/license.db*, the *license database file*. (The administration directory is */var/adm* on some platforms.)
- **On each installation host** — placing the name of the license server host in text file */usr/adm/atria/config/license_host*

In order to use ClearCase, each user must have a *license*, which grants the user the privilege to use ClearCase commands and data on any number of hosts in the network. If no more licenses are available at a particular time, a user with a higher *license priority* automatically *bumps* (replaces) a lower priority user. The highest priority level is 1.

Use the *clearlicense* utility to determine your current licensing status.

LICENSE DATABASE FILE FORMAT

The license database file contains several kinds of lines. A line can define a multiuser license, specify users' license priorities, or enable auditing of licensing activity.

All lines in the license database file must be terminated with a <NL> character.

License Set Definition Lines

When you first obtain ClearCase, your vendor provides you with a single line of text, which defines a certain number of licenses. This line must be entered, *exactly* as provided, in the license database file on the license server host.

Most licenses are locked to their particular license server host. You cannot move the *license.db* file to any other host without invalidating the license. If the *vendor* field is `TEMPORARY`, you can move the *license.db* file around the network, to any ClearCase installation host.

The license database file can contain any number of `-license` lines. All the lines are effectively combined into a single license; the maximum numbers accumulate to determine the total number of license slots. Alternatively, it may be better to split licenses among two or more license servers. This increases ClearCase availability: if one license server host goes down, the licenses on the other license server hosts can still be used.

User Priority Lines

The license database file can contain any number of `-user` lines, each of which specifies one or more users (by name or by numeric ID). All these lines are effectively concatenated into a single license priority list. The first user on the list has the highest priority; each successive user has a lower priority. Users not listed at all can still use ClearCase, but they all share the lowest priority.

Excluded User Lines

The license database file can contain any number of `-nuser` lines, each of which specifies one or more users (by name or by numeric ID). The specified users cannot obtain a license and, thus, are completely forbidden from using ClearCase.

`-user` and `-nuser` lines can be intermixed. If a user is named in both kinds of line, the first entry wins.

Audit-Enable Line

A line consisting of the single word `-audit` enables auditing of license activity. An audit message is logged to */usr/adm/atria/log/albd_log* when:

- a user is granted a new license
- a user is denied a license because all licenses are in use
- a user entered a `clearlicense -release` command. The success/failure of the command is logged, also.

Timeout Line

By default, a license granted to a user expires in 60 minutes if the user does not enter any additional ClearCase commands. A `-timeout` line changes the expiration interval to the specified number of minutes. The minimum interval is 30 minutes; there is no maximum interval.

EXAMPLES

- The following line defines a ClearCase license for a maximum of 10 active users. The license expires on November 16, 1992.

```
-license ClearCase ATRIA *.10 19921116 2adde977.1360cb11.02
```

- The following lines define licenses that accommodate a total of 13 active users. User *adm* is assigned the highest priority, *smith* the next highest, and *akp* the next highest. The 10-user license expires at the beginning of November 16, 1992, but the 3-user license has no expiration date.

```
-license ClearCase ATRIA *.10 19921116 2adde977.1360cb11.02
-license ClearCase ATRIA *.3 NONE 2adde9b9.682410da.02
-user adm
-user smith akp
```

SEE ALSO

clearlicense, albd_server

NAME lockmgr – VOB database access arbitrator

SYNOPSIS

invoked by ClearCase startup script

DESCRIPTION

Each VOB host runs one database lock manager process, *lockmgr*. This process arbitrates transaction requests to all the VOB databases on that host, from ClearCase client programs throughout the network. The calling program polls *lockmgr*, which either grants or prohibits access to the requested data. If the data is available, the transaction proceeds immediately: the data is read or written, and output is returned to the calling program. If the data is unavailable ("locked" because another caller has been granted "write" access to the data), the caller waits until *lockmgr* grants it access to the data.

The ClearCase startup script invokes a *lockmgr* process at system startup time. This command sets the number of "file slots" to 256, enabling the host to accommodate up to 36 concurrently-active VOBs. To change this limit, modify the value of the `–f` option in the command that starts *lockmgr*:

```
${ATRIA}/etc/lockmgr ...
```

Lock Manager Socket

lockmgr creates a socket, */tmp/.A/almd*, when it begins execution. It communicates with calling processes through this socket. To reduce the likelihood of accidental deletion, the socket is created within a sub-directory of */tmp*, and is owned by the *root* user.

The *root* user's *crontab*(1) should be examined, and modified if necessary, to ensure that the socket for a long-lived *lockmgr* process is not deleted accidentally. For example, the following *find*(1) command includes a "not a socket" clause:

```
find /tmp  ! \( -type s \) -exec rm -f {} \;
```

ERROR LOG

The *lockmgr* sends warning and error messages to */usr/adm/atria/log/lockmgr_log*.

SEE ALSO

albd_server, db_server, init_ccase

NAME makefile_ccase – target description file for clearmake builds

SYNOPSIS

file(s) read by ClearCase build program, 'clearmake', containing instructions for creating derived objects

DESCRIPTION

This manual page discusses the format of *target description files*, or *makefiles*, processed by the ClearCase build program, *clearmake*. This is not a complete, rigorous description of makefile syntax; rather, it is a discussion of differences and ClearCase-specific extensions.

MAKEFILE FORMAT

A *makefile* contains a sequence of entries, each of which specifies some dependencies and *build scripts* of commands to be executed. A makefile can also contain *make macro* definitions and build directives (*special targets*.)

- **Target/dependencies line** — The first line of an entry is a white-space-separated, non-null list of *targets*, followed by a colon (`:`) or a double colon (`::`), and a (possibly empty) list of *dependencies*. Both targets and dependencies may contain ClearCase pathname patterns. (See the *wildcards_ccase* manual page.)

  The list of dependencies often need not include source objects, such as header files — such dependencies are detected automatically by *clearmake*. But the list must include build-order dependencies — for example, object modules and libraries that must be built before executables.

- **Build script** — Text following a semicolon (`;`) on the same line, and all subsequent lines that begin with a `<Tab>` character, constitute a *build script*: a set of shell commands to be executed. A shell command can be continued onto the next text line with a `\<NL>` sequence. Any line beginning with a pound sign (#) character is a comment.

  A build script ends at the first non-empty line that does not begin with a `<Tab>` or pound sign (#) character; this begins a new target/dependencies line or a make macro definition.

  Note that *clearmake* always completely eliminates a `\<NL>` sequence, even in its compatibility modes. Some other *make* programs sometimes preserve such a sequence — for example, in a *sed*(1) "insert" command:

  ```
  target: depdcy
          sed -e '/xxx=0/i\
                     yyy=xxx;' depdcy > target
  ```

  Build scripts should use standard pathnames only — not view-extended pathnames, nor version-extended pathnames.

  Executing a build script "updates the target", and is termed a *target rebuild*. The shell commands in a build script are executed one at a time, each in its own subshell.

- **Make macro** — A *make macro* is an assignment of a character-string value to a simple name. By convention, all letters in the name are uppercase (for example, *CFLAGS*).

- **Special targets** — A line that begins with a dot (`.`) character is a *special target*, which acts as a directive to the build utility, *clearmake*.

RESTRICTIONS

clearmake does not have any built-in rules to support the use of SCCS files in a makefile. *clearmake* does not support the use of standard input (–) as a makefile.

LIBRARIES

If a target or dependency name contains parentheses, it is assumed to be an archive (library) created by *ar*(1). For example:

```
lib.a : lib.a(mod1.o) lib.a(mod2.o)
```

The string within parentheses refers to a member (object module) within the library. Use of function names within parentheses is not supported. Thus, `lib.a(mod1.o)` refers to an archive that contains object module *mod1.o*. The expression `lib.a(mod1.o mod2.o)` is not valid.

Inference rules for archive libraries have this form:

.*sfx***.a**

… where *sfx* is the file name suffix from which the archive member is to be made.

ClearCase does not support incremental updating of derived objects. Thus, the way in which *clearmake* handles archive construction differs from other *make* variants. For more on this topic, see the "Makefile Optimization" chapter in the *ClearCase User´s Manual*.

COMMAND ECHOING AND ERROR HANDLING

You can control the echoing of commands and the handling of errors that occur during command execution on a line-by-line basis, or on a global basis.

You can prefix any command with one or two characters, as follows:

– Causes *clearmake* to ignore any errors during execution of the command. By default, an error causes *clearmake* to terminate.

The command-line option `–i` suppresses termination-on-error for *all* command lines.

`@` Suppresses display of the command line. By default, *clearmake* displays each command line just before executing it.

The command-line option `–s` suppresses display of *all* command lines. The `–n` option does the opposite — commands are displayed but not executed.

`–@` `@–` These two prefixes combine the effect of – and `@`.

The `–k` option provides for partial recovery from errors. If an error occurs, execution of the current target (that is, the set of commands for the current target) is abandoned, but execution continues on other targets that do not depend on that target.

BUILT-IN RULES

Suffixes and their associated rules in the makefile override any identical suffixes in the built–in rules. *clearmake* reads built-in rules from the file */usr/atria/etc/builtin.mk*.

INCLUDE FILES

If a line in a makefile starts with the string `include` or `sinclude` followed by white space (at least one `<Space>` or `<Tab>` character), the rest of the line is assumed to be a file name. (This name can contain macros.) The contents of the file are effectively placed at the current location in the makefile.

For `include`, a fatal error occurs if the file is not readable. For `sinclude`, a non-readable file is silently ignored. Include files may be nested to a maximum of 17 levels.

MAKE MACROS

A *macro definition* takes this form:

macro_name **=** *string*

Macros can appear in the makefile, on the command line, or in a *build options specification* file. (See the *clearmake.options* manual page.)

Macro definitions require no quotes or delimiters, except for the equal sign(=) character, which separates the macro name from the value. Leading and trailing white space characters are stripped. Lines can be continued using a `\<NL>` sequence; this sequence and all surrounding white space is effectively converted to a single `<Space>` character. *macro_name* cannot include white space, but *string* can — it includes all characters up to an unescaped `<NL>` character.

clearmake performs macro substitution whenever it encounters either of the following in the makefile:

$(*macro_name***)**

$(*macro_name***:***subst1***=***subst2***)**

It substitutes *string* for the macro invocation. In the latter two forms, it performs an additional substitution within *string*: all occurrences of *subst1* at the end of a word within *string* are replaced by *subst2*. If *subst1* is empty, *subst2* is appended to each word in the value of *macro_name*; If *subst2* is empty, *subst1* is removed from each word in the value of *macro_name*.

Example:

```
% cat Makefile
C_SOURCES = one.c two.c three.c four.c
test:
      echo "OBJECT FILES are: $(C_SOURCES:.c=.o)"
      echo "EXECUTABLES are:  $(C_SOURCES:.c=)"
% clearmake test
OBJECT FILES are: one.o two.o three.o four.o
EXECUTABLES are:  one two three four
```

INTERNAL MACROS

clearmake maintains these macros internally, which are useful for writing rules for building targets:

$* (defined only for inference rules) The file name part of the inferred dependency, with the suffix deleted.

$@ The full target name of the current target.

$< (defined only for inference rules) The file name of the implicit dependency.

$? (defined only when explicit rules from the makefile are evaluated) The list of dependencies that are out-of-date with respect to the target. When configuration lookup is enabled (default), it expands to the list of all dependencies. When a dependency is an archive library member of the form `lib(file.o)`, the name of the member, *file.o*, appears in the list.

$$@ (defined only on dependency lines in makefiles) The file name of the current target. In the example below, the dependency is translated at makefile-parse time, first to the string `cat.c`, then to the string `dd.c`:

```
cat dd: $$@.c
```

$% (defined only when the target is an archive library member) For a target of the form `lib(file.o)`, **$@** evaluates to `lib` and **$%** evaluates to the library member, *file.o*.

MAKEFILE

During makefile parsing, this macro expands to the pathname of the current makefile. After makefile parsing is complete, it expands to the pathname of the last makefile that was parsed. This holds only for top-level makefiles, not for included makefiles or for built-in rules.

Use this macro as an explicit dependency to include the version of the makefile in the CR produced by a target rebuild. For example:

```
supersort: main.o sort.o cmd.o $(MAKEFILE)
        cc -o supersort ...
```

VPATH MACRO

The VPATH macro specifies a search path for targets. Its value can be one directory pathname, or a colon-separated list of directory pathnames. *clearmake* searches the directories on the VPATH when it fails to find a target in the current working directory.

Configuration lookup is VPATH-sensitive when qualifying makefile dependencies (explicit dependencies in the makefile). Thus, if a newer version of a dependent file appears in a directory on the search path *before* the pathname in the CR (the version used in the previous build), *clearmake* rejects the previous build, and rebuilds the target with the new file.

The VPATH setting may affect the expansion of internal macros, such as `$<`.

SPECIAL TARGETS

clearmake supports these special targets in makefiles:

.DEFAULT If a file must be built, but there are no explicit commands or relevant built-in rules to build it, the commands associated with this target are used (if it exists).

.PRECIOUS
Dependents of this target will not be removed when a `QUIT` character (typically, `<Ctrl-\>`) or an `INTR` character (typically, `<Ctrl-C>`) is typed.

.NOTPARALLEL
Disables parallel building for the current makefile. It does not affect lower-level builds in a recursive make (unless present in the makefiles for those builds).

.SILENT Same effect as the `-s` option.

.IGNORE Same effect as the `-i` option.

NOTE: The following special targets can be used either in the makefile itself or in a *build options specification* file. (See the *clearmake.options* manual page.)

.NO_CONFIG_REC : *tgt …*
The specified targets will be built as if the `-F` option was specified: modification time is used for build avoidance, and no CRs or derived objects are created. You might use this target in a build options specification file to allow incremental updating of *ar* archives.

.NO_CMP_SCRIPT : *tgt …*
The specified targets will be built as if the `-O` option was specified: build scripts are not compared during configuration lookup. This is useful when different makefiles (and, hence, different build scripts) are regularly used to build the same target.

.NO_WINK_IN : *tgt …*
The specified targets will be built as if the `-V` option was specified: configuration lookup is restricted to the current view.

.NO_CMP_NON_MF_DEPS : *tgt …*
The specified targets will be built as if the `-M` option was specified: if a dependency is not explicitly declared in the makefile, it is not used in configuration lookup.

SEE ALSO

bldhost, bldserver.control, clearmake, clearmake.options, wildcards_ccase

NAME mount_ccase – mount/unmount commands for VOBs and the viewroot directory

DESCRIPTION

MVFS file systems, VOBs and the *viewroot* directory, are mounted and unmounted by the standard *mount*(1M) and *umount*(1M) commands. ClearCase also includes a utility, *clearcase_domounts*, for handling this work. The details are architecture-specific — consult the following manual pages:

SunOS-4	mount_sun4
SunOS-5	mount_sun5
HPUX-9	mount_hpx9
IRIX-5	mount_irx5
OSF/1 V2	mount_osf1

SEE ALSO

exports_ccase, mount_ccase, filesys_ccase
mount(1M), umount(1M), mountall(1M) [some architectures]

NAME mount_hpx9 – ClearCase-specific mount utility: mount_mvfs (HPUX-9)

SYNOPSIS

/etc/mount

replaced during ClearCase installation
invoked as needed by cleartool's 'mount' subcommand

DESCRIPTION

This manual page describes the mechanisms that mount VOBs as file systems of type MVFS (the ClearCase *multiversion file system*). Also included is a description of "obsolete" ClearCase Release 1.1.x functionality that continues to be supported in the current release, but will be withdrawn in a future release.

Automatic VOB Activation at System Startup. At system startup, the ClearCase startup script, */etc/rc.atria*, issues a `cleartool mount -all` command. This activates on the local host all the VOBs that are registered as *public* in the (local host's *network region* of the) ClearCase *storage registry*. During this procedure, the program */etc/mount* performs the actual work of mounting the VOB as a file system of type MVFS. ClearCase installation replaces the standard */etc/mount* program with a ClearCase-supplied compiled program, which knows how to mount file systems of type `mvfs`. The original program is renamed to *mount.9.0*.

No comparable manipulation of the */etc/umount* command is required or performed.

VOB Activation after System Startup. After system startup, a *cleartool mount* command can be used to (re)activate any VOB that is listed in the storage registry.

- The *root* user can activate any VOB in this way.
- A non-*root* user can activate any *public* VOB, or any *private* VOB owned by that user.

Automatic VOB Deactivation at System Shutdown. At system shutdown, the script is invoked with the `stop` option to execute the ClearCase shutdown procedure. As part of this procedure, a `cleartool umount -all` command deactivates on the local host all the VOBs that are currently active there. This command invokes the standard *umount*(1M) utility.

Individual VOB Deactivation. While ClearCase is running, a *cleartool umount* command can be used to deactivate any mounted VOB:

- The *root* user can deactivate any VOB in this way.
- A non-*root* user can deactivate any *public* VOB, or any *private* VOB owned by that user.

USE OF FILE SYSTEM TABLE: OBSOLETE, BUT STILL SUPPORTED

The *root* user can invoke the script */usr/atria/etc/clearcase_domounts* to mount all VOBs listed in a particular file or NIS map. In Release 1.1.x, this script was used by the ClearCase startup script to process the ClearCase-specific file system table, */etc/fstab.mfs*. (If your host had such a file, installation of this release will have renamed it to */etc/fstab.mvfs*.) See the *filesys_ccase* manual page for descriptions of the "obsolete" ClearCase-specific file system table and NIS map, and the *clearcase_domounts* script.

NOTES

The *mount_mvfs* program should never be invoked explicitly.

SEE ALSO

cleartool subcommands: mount, umount
exports_ccase, filesys_ccase, mount_ccase
mount(1M), umount(1M)

NAME mount_irx5 – ClearCase-specific mount utility: mount_mvfs (IRIX-5)

SYNOPSIS

/usr/etc/mount_mvfs

invoked as needed by cleartool's 'mount' subcommand

DESCRIPTION

This manual page describes the mechanisms that mount VOBs as file systems of type MVFS (the ClearCase *multiversion file system*). Also included is a description of "obsolete" ClearCase Release 1.1.x functionality that continues to be supported in the current release, but will be withdrawn in a future release.

Automatic VOB Activation at System Startup. At system startup, the ClearCase startup script, */etc/init.d/atria*, issues a `cleartool mount -all` command. This activates on the local host all the VOBs that are registered as *public* in the (local host's *network region* of the) ClearCase *storage registry*. During this procedure, the program */usr/etc/mount_mvfs* performs the actual work of mounting the VOB as a file system of type MVFS. (This is actually a symbolic link to */usr/atria/etc/mount_mvfs*.)

VOB Activation after System Startup. After system startup, a *cleartool mount* command can be used to (re)activate any VOB that is listed in the storage registry.

- The *root* user can activate any VOB in this way.
- A non-*root* user can activate any *public* VOB, or any *private* VOB owned by that user.

Automatic VOB Deactivation at System Shutdown. At system shutdown, the script is invoked with the `stop` option to execute the ClearCase shutdown procedure. As part of this procedure, a `cleartool umount -all` command deactivates on the local host all the VOBs that are currently active there. This command invokes the standard *umount*(1M) utility.

Individual VOB Deactivation. While ClearCase is running, a *cleartool umount* command can be used to deactivate any mounted VOB:

- The *root* user can deactivate any VOB in this way.
- A non-*root* user can deactivate any *public* VOB, or any *private* VOB owned by that user.

USE OF FILE SYSTEM TABLE: OBSOLETE, BUT STILL SUPPORTED

The *root* user can invoke the script */usr/atria/etc/clearcase_domounts* to mount all VOBs listed in a particular file or NIS map. In Release 1.1.x, this script was used by the ClearCase startup script to process the ClearCase-specific file system table, */etc/fstab.mfs*. (If your host had such a file, installation of this release will have renamed it to */etc/fstab.mvfs*.) See the *filesys_ccase* manual page for descriptions of the "obsolete" ClearCase-specific file system table and NIS map, and the *clearcase_domounts* script.

NOTES

The *mount_mvfs* program should never be invoked explicitly.

SEE ALSO

cleartool subcommands: mount, umount
exports_ccase, filesys_ccase, mount_ccase
mount(1M), umount(1M)

NAME mount_osf1 – ClearCase-specific mount utility: mount_mvfs (OSF/1)

SYNOPSIS

/sbin/mount_mvfs

invoked as needed by cleartool's 'mount' subcommand

DESCRIPTION

This manual page describes the mechanisms that mount VOBs as file systems of type MVFS (the ClearCase *multiversion file system*). Also included is a description of "obsolete" ClearCase Release 1.1.x functionality that continues to be supported in the current release, but will be withdrawn in a future release.

Automatic VOB Activation at System Startup. At system startup, the ClearCase startup script, */sbin/init.d/atria*, issues a `cleartool mount -all` command. This activates on the local host all the VOBs that are registered as *public* in the (local host's *network region* of the) ClearCase *storage registry*. During this procedure, the program */sbin/mount_mvfs* performs the actual work of mounting the VOB as a file system of type MVFS. (This is actually a symbolic link to */usr/atria/etc/mount_mvfs*.)

VOB Activation after System Startup. After system startup, a *cleartool mount* command can be used to (re)activate any VOB that is listed in the storage registry.

- The *root* user can activate any VOB in this way.
- A non-*root* user can activate any *public* VOB, or any *private* VOB owned by that user.

Automatic VOB Deactivation at System Shutdown. At system shutdown, the script is invoked with the `stop` option to execute the ClearCase shutdown procedure. As part of this procedure, a `cleartool umount -all` command deactivates on the local host all the VOBs that are currently active there. This command invokes the standard *umount*(1M) utility.

Individual VOB Deactivation. While ClearCase is running, a *cleartool umount* command can be used to deactivate any mounted VOB:

- The *root* user can deactivate any VOB in this way.
- A non-*root* user can deactivate any *public* VOB, or any *private* VOB owned by that user.

USE OF FILE SYSTEM TABLE: OBSOLETE, BUT STILL SUPPORTED

The *root* user can invoke the script */usr/atria/etc/clearcase_domounts* to mount all VOBs listed in a particular file or NIS map. See the *filesys_ccase* manual page for descriptions of the "obsolete" ClearCase-specific file system table and NIS map, and the *clearcase_domounts* script.

NOTES

The *mount_mvfs* program should never be invoked explicitly.

SEE ALSO

cleartool subcommands: mount, umount
exports_ccase, filesys_ccase, mount_ccase
mount(1M), umount(1M)

NAME mount_sun4 – ClearCase-specific mount utility: mount_mvfs (SunOS-4)

SYNOPSIS

/usr/etc/mount_mvfs

invoked as needed by cleartool's 'mount' subcommand

DESCRIPTION

This manual page describes the mechanisms that mount VOBs as file systems of type MVFS (the ClearCase *multiversion file system*). Also included is a description of "obsolete" ClearCase Release 1.1.x functionality that continues to be supported in the current release, but will be withdrawn in a future release.

Automatic VOB Activation at System Startup. At system startup, the ClearCase startup script, */etc/rc.atria*, issues a `cleartool mount -all` command. This activates on the local host all the VOBs that are registered as *public* in the (local host's *network region* of the) ClearCase *storage registry*. During this procedure, the program */usr/etc/mount_mvfs* performs the actual work of mounting the VOB as a file system of type MVFS. (This is actually a symbolic link to */usr/atria/etc/mount_mvfs*.)

VOB Activation after System Startup. After system startup, a *cleartool mount* command can be used to (re)activate any VOB that is listed in the storage registry.

- The *root* user can activate any VOB in this way.
- A non-*root* user can activate any *public* VOB, or any *private* VOB owned by that user.

Automatic VOB Deactivation at System Shutdown. At system shutdown, the script is invoked with the `stop` option to execute the ClearCase shutdown procedure. As part of this procedure, a `cleartool umount -all` command deactivates on the local host all the VOBs that are currently active there. This command invokes the standard *umount*(1M) utility.

Individual VOB Deactivation. While ClearCase is running, a *cleartool umount* command can be used to deactivate any mounted VOB:

- The *root* user can deactivate any VOB in this way.
- A non-*root* user can deactivate any *public* VOB, or any *private* VOB owned by that user.

USE OF FILE SYSTEM TABLE: OBSOLETE, BUT STILL SUPPORTED

The *root* user can invoke the script */usr/atria/etc/clearcase_domounts* to mount all VOBs listed in a particular file or NIS map. In Release 1.1.x, this script was used by the ClearCase startup script to process the ClearCase-specific file system table, */etc/fstab.mfs*. (If your host had such a file, installation of this release will have renamed it to */etc/fstab.mvfs*.) See the *filesys_ccase* manual page for descriptions of the "obsolete" ClearCase-specific file system table and NIS map, and the *clearcase_domounts* script.

NOTES

The *mount_mvfs* program should never be invoked explicitly.

SEE ALSO

cleartool subcommands: mount, umount
exports_ccase, filesys_ccase, mount_ccase
mount(1M), umount(1M)

NAME mount_sun5 – ClearCase-specific mount utility: mount_mvfs (SunOS-5)

SYNOPSIS

/usr/lib/fs/mvfs/mount

invoked as needed by cleartool's 'mount' subcommand

DESCRIPTION

This manual page describes the mechanisms that mount VOBs as file systems of type MVFS (the ClearCase *multiversion file system*). Also included is a description of "obsolete" ClearCase Release 1.1.x functionality that continues to be supported in the current release, but will be withdrawn in a future release.

Automatic VOB Activation at System Startup. At system startup, the ClearCase startup script, */etc/init.d/atria*, issues a `cleartool mount -all` command. This activates on the local host all the VOBs that are registered as *public* in the (local host's *network region* of the) ClearCase *storage registry*. During this procedure, the program */usr/lib/fs/mvfs/mount* performs the actual work of mounting the VOB as a file system of type MVFS. (This is actually a symbolic link to */usr/atria/etc/mount_mvfs*.)

VOB Activation after System Startup. After system startup, a *cleartool mount* command can be used to (re)activate any VOB that is listed in the storage registry.

- The *root* user can activate any VOB in this way.
- A non-*root* user can activate any *public* VOB, or any *private* VOB owned by that user.

Automatic VOB Deactivation at System Shutdown. At system shutdown, the script is invoked with the `stop` option to execute the ClearCase shutdown procedure. As part of this procedure, a `cleartool umount -all` command deactivates on the local host all the VOBs that are currently active there. This command invokes the standard *umount*(1M) utility.

Individual VOB Deactivation. While ClearCase is running, a *cleartool umount* command can be used to deactivate any mounted VOB:

- The *root* user can deactivate any VOB in this way.
- A non-*root* user can deactivate any *public* VOB, or any *private* VOB owned by that user.

USE OF FILE SYSTEM TABLE: OBSOLETE, BUT STILL SUPPORTED

The *root* user can invoke the script */usr/atria/etc/clearcase_domounts* to mount all VOBs listed in a particular file or NIS map. In Release 1.1.x, this script was used by the ClearCase startup script to process the ClearCase-specific file system table, */etc/fstab.mfs*. (If your host had such a file, installation of this release will have renamed it to */etc/fstab.mvfs*.) See the *filesys_ccase* manual page for descriptions of the "obsolete" ClearCase-specific file system table and NIS map, and the *clearcase_domounts* script.

NOTES

The *mount_mvfs* program should never be invoked explicitly.

SEE ALSO

cleartool subcommands: mount, umount
exports_ccase, filesys_ccase, mount_ccase
mount(1M), umount(1M)

NAME mvfscache – control and monitor MVFS caches

SYNOPSIS

- Determine cache status:

  /usr/atria/etc/mvfscache [*cache_name*]

- Control cache operation:

  /usr/atria/etc/mvfscache { **–e** *cache_list* | **–d** *cache_list* | **–f** *cache_list* }

DESCRIPTION

mvfscache maintains a host's *MVFS caches*, which are used to optimize file system performance. The *root* user can display or change a cache's enabled/disabled status. Any user can *flush* a cache. However, this utility is not intended for general use. It is intended primarily to help ClearCase engineering and Customer Support personnel diagnose problems with the MVFS.

OPTIONS AND ARGUMENTS

Determining Cache Status. With no options or arguments, *mvfscache* displays the enabled/disabled status of all MVFS caches. If you don't use any of the options, but specify a cache name as an argument, *mvfscache* does not display any output; it just returns an appropriate exit status:

0	specified cache is enabled
1	specified cache is disabled

Controlling Cache Operation. Use one of the following options to control a cache, or a set of caches.

–e *cache-list* (must be *root*) Enables the specified caches and cache-related behaviors. The *cache-list* can include any number of the following keywords; the list must be comma-separated, with no white space.

attr	*attribute cache* — caches *stat*(2) records for recently accessed objects.
name	*name cache* — caches name lookup translations for recently accessed files and directories.
noent	*name-not-found cache* — (a portion of the name cache) caches recent name lookups that returned ENOENT.
rvc	*VOB root version cache* — caches VOB mount point data for each view.
slink	*symbolic link text cache* — caches the contents of recently accessed symbolic links.

The remaining keywords enable cache-related behaviors, rather than actual caches:

cto	*close-to-open consistency* — force a *stat* operation to the *view_server* on every OS open operation.
autocd	*automatic cd* — automatically *cd*'s the user to a new version of the current working directory, if one is checked in to the current view.

–d *cache-list*

(must be *root*) Disables the specified caches and cache-related behaviors. The syntax is the same as for –e.

–**f** *cache-list* Flushes the specified cache(s). The *cache-list* can include any number of the following keywords; the list must be comma-separated, with no white space.

mnode *mnode freelist cache* — flushes attrs, "slink text", open freelist files, and mnode storage for all freelist mnodes.

name name cache

rvc VOB root version cache

EXAMPLES

- Determine the status of all caches.

```
% /usr/atria/etc/mvfscache
Attr:   on
Name:   on
Noent:  on
Rvc:    on
Slink:  on
Cto:    on
Autocd: on
```

- Clear busy mount points, to prepare for unmounting VOBs.

```
% /usr/atria/etc/mvfscache –f mnode
```

- Enable the *name* and *noent* caches:

```
% /usr/atria/etc/mvfscache –e name,noent
```

SEE ALSO

mvfslog, mvfsstat, mvfsstorage, mvfstime, mvfsversion
csh(1), init_ccase, stat(2)

NAME mvfslog – set or display MVFS console error logging level

SYNOPSIS

/usr/atria/etc/mvfslog [**none** | **error** | **warn** | **info** | **stale** | **debug**]

DESCRIPTION

Only the 'root' user can use this command to change the error logging level.

Sets or displays the verbosity level for MVFS console error logging. The initial setting is `error`: only RPC errors and actual MVFS errors are logged; warnings and diagnostics are suppressed.

OPTIONS AND ARGUMENTS

Default: Displays the current error logging level. Use one of the following keywords to specify a new level; `none` is the least verbose; `debug` is the most verbose.

none RPC errors only.

error MVFS errors are logged (default setting).

warn MVFS warnings are logged.

info MVFS diagnostics on some expected errors are logged.

stale MVFS diagnostics related to ESTALE errors are logged.

debug Verbose information on many expected errors.

SEE ALSO

mvfscache, mvfsstat, mvfsstorage, mvfstime, mvfsversion

NAME mvfsstat – list MVFS statistics

SYNOPSIS

/usr/atria/etc/mvfsstat [**–chilrvV**] [*time*] [*count*]

DESCRIPTION

Displays MVFS usage and operating statistics, including cumulative statistics on MVFS cache usage, *rpc* statistics, cleartext I/O counts, *vnode* operation counts, and *VFS* operation counts. This data is useful for evaluating file system performance and determining whether MVFS cache sizes require adjustment.

MVFS CACHE STATISTICS

The `–c` option reports on the usage of the host's MVFS caches. This report is cumulative, covering the entire period since the operating system was last restarted. The following example covers 16-day period:

```
---------------- Mon Mar  7 11:50:06 1994 --------------------
dnlc:   527187  453323( 86.0%) hit      23728 dot 234884 dir 118954 reg 75757 noent
                73864 ( 14.0%) miss     15097 tl 246 gen 233 ev 1803 timeout 0 novp
                60935 ( 11.6%) add      26544 tlmiss 0 unlkmiss 0 noop
                                        2167 dir 30544 reg 28224 noent
                                        203 addbh 0 addbhinvar
                                        40442 hitbhinvar 203 missbh
                                        12038 change 0 remove
                                        2485 flushvp 12 flush 27088 ents
attr:   363967      295000( 81.1%) hit  68967 (31243cto+8237ev) miss
                                        178658 upd 15866+30094 mod 268 vmod 101225 aud
slink:    5811        5326( 91.7%) hit  485 miss
rvc:    191570      189309( 98.8%) hit  2261 miss
```

The following sections describe the particular statistics that are useful in tuning MVFS performance on a ClearCase client host.

Directory Name Lookup Cache (dnlc)

The `dnlc` section reports on usage of a name-lookup cache that maps pathnames to ClearCase-internal identifiers. Note that the value precedes the keyword — for example, `23728 dot` means that the reported value of the "dot" statistic is 23728.

Cache Hits. The `hit` line reports on cache hits:

`dot` Number of times the current working directory was looked up (always a cache hit)

`dir` Number of times a directory object was hit in the cache.

`reg` Number of times a file object was hit in the cache.

`noent` Number of times a cached ENOENT return was found.

This cache has low hit rates (around 50%) for activities that walk a large tree — for example, a *find* command, or a recursive *clearmake* that examines many files and determines that nothing needs to be built.

Cache Misses. The `miss` line reports on total cache misses. In some cases a cache miss occurs "normally", just because there was no entry in the cache; in other cases, there is a "special" reason for the cache miss:

`tl` Attempted to lookup a name which was too long (> 24 characters).

`gen` Misses due to MVFS-internal readjustments.

`ev` Misses due to a significant VOB event (such as version labeling or branch creation) or because the *view_server*'s cache is too small.

`timeout` Misses due to the entry "timing out". This is most common on the ENOENT portion of the cache, where there is no object to tie VOB events to.

novp Misses due to vnode recycling before completion of cache lookup.

Cache Additions. The `add` line report on cache misses that occurred because a new entry was being added to the cache. The additions are categorized as directory entries (*dir*), file entries (*reg*), and ENOENT entries (*noent*).

The *flushvp* statistic reports the number of times a specific object flushed from the cache; the *flush* statistic reports the number of times the whole cache was flushed.

Attribute Cache

The `attr` section reports on usage of a cache of *stat*(2) returns. This cache generally has hit rates comparable to that for the directory name lookup cache.

Symlink Text Cache

The `slink` section reports on usage of a cache of symbolic link texts (both VOB symbolic links and view-private symbolic links). This number has a small effect on performance.

Root Version Cache

The `rvc` section reports on usage of a specialized cache for the version of a VOB mount point. This number has a small effect on performance.

OPTIONS AND ARGUMENTS

time Time in seconds between samples. Display deltas on each sample. If you omit this option, only the absolute values of all information are printed.

count Number of samples. If omitted, defaults to "infinite".

−c Display statistics for the MVFS caches, as described in "MVFS Cache Statistics" above.

−r Display MVFS remote-procedure-call (RPC) statistics. These statistics include both counts and "real time waited". Real-time waited may be greater than 100% of a sample period in two cases:

- when an operation took longer to complete than the sample period; for example, 60 seconds of "wait time" is recorded in a 30-second sample
- multiple processes are waiting at the same time

In general, real-time percentages are meaningful only when a single process is accessing a VOB.

–i Display cleartext I/O counts and wait times.

–v Display counts of *vnode* operations.

–V Display counts of *vfs* operations.

–h Display an RPC histogram. Cleartext fetch RPCs are tallied separately from all other RPCs.

–l Add more detail to the statistics generated by `–r`, `–i`, `–v`, and/or `–V`, by providing a breakdown by individual operations.

SEE ALSO

mfscache, mvfslog, mvfsstorage, mvfstime, mvfsversion

NAME mvfsstorage – list data container pathname for MVFS file

SYNOPSIS

/usr/atria/etc/mvfsstorage *pname* …

DESCRIPTION

mvfsstorage lists the pathname of an MVFS file's data container.

The contents of an *MVFS file* are stored in a *data container*. There are several cases:

1. Versions of file elements are stored in data containers located in a VOB *source storage pool*.

   NOTE: *mvfsstorage* does not list these data containers; use `cleartool dump` instead.

2. Recently-accessed versions of *text_file* and *compressed_text_file* elements may be cached in data containers located in a VOB *cleartext storage pool*.

3. View-private files (including checked-out versions and unshared derived objects) are stored in data containers located in a view's private storage area.

mvfsstorage is intended for use in finding discrepancies in UNIX access rights between the view and the underlying MVFS storage. Usually, such discrepancies occur as a result of *set–UID root*, or creating files as *root*. If you encounter a permissions error that seems unfounded, you should run this utility as a diagnostic.

This command takes one or more pathnames as its argument. These should be the names of files whose pathnames are under a VOB-tag (an *MVFS object*). For directories and non-MVFS objects, *mvfsstorage* simply echos the pathname(s) you give it.

EXAMPLES

- For a view-private file, compare view-level ownership and permissions against those on the file's underlying storage location.

```
% ls –l unixV7 ‘mvfsstorage unixV7‘
-rwxrwxrwx 1 nobody 65534  2210032 May 12 09:33 /net/myhost/home/myview/
        .s/0008.VOB/016D.2E2F.unixV7*
-rwxrwxrwx 1 root   sys    2210032 May 12 09:33 unixV7*
```

SEE ALSO

mfscache, mvfslog, mvfsstat, mvfstime, mvfsversion

NAME mvfstime – list MVFS timing statistics for a command

SYNOPSIS

/usr/atria/etc/mvfstime [**–i**] [**–c**] [**–r**] [**–v**] [**–V**] [**–l**] [**–h**] *command* [*args*]

DESCRIPTION

Executes a command and sends to *stderr* a report consisting of:

- standard UNIX timing statistics
- MVFS usage statistics, similar to those generated by *mvfsstat*.

Use this command to perform timing experiments for applications running in a ClearCase environment.

See the *mvfsstat* manual page for an explanation of MVFS statistics. See the *csh*(1) manual page for information on UNIX statistics.

OPTIONS AND ARGUMENTS

See the *mvfsstat* manual page for a description of the command-line options.

EXAMPLES

- Generate timing statistics for an invocation of the *make* program.

```
% mvfstime –iclr make afprint
linking afprint
   rm -f afprint
   cc -o afprint  afprint.o   -g  -L/vobs/atria/sgi4/pvtlib \
              -L/vobs/atria/sgi4/lib  -ltbs -lks -lm -lcurses -lsun

----------------- Wed May  2 17:33:36 1992 ---------------------
time:    0.8u  2.1s  0:10 28% 215+62io 9pf+0w
dnlc:      124  74   +  49( 99.2%) hit  1 (0tl+0vm+0bh) miss
                                        1 ch 0 rm 0/0 purge
attr:      281         271( 96.4%) hit  10 (10cto+0ev) miss
                                        12 upd 1+1 mod 0 vmod
rvc:        56          56(100.0%) hit  0 miss
--------clrio-------calls---c/s------rt--rt/call
get/create              1  0.10     0.1  0.080   1%
rdwr                  365 35.70     1.7  0.005  17%
clrio total:          366 35.80     1.8         17%
----------rpc-------calls---c/s------rt--rt/call
getattr                10  0.98     3.1  0.310  30%
setattr                 1  0.10     0.0  0.020   0%
lookup                  1  0.10     0.0  0.030   0%
create                  1  0.10     0.1  0.080   1%
readdir                 4  0.39     0.1  0.035   1%
rpc total:             17  1.66     3.4         33%  0 retrans
```

SEE ALSO

mfscache, mvfslog, mvfsstat, mvfsstorage, mvfsversion
csh(1)

NAME mvfsversion – list MVFS version string

SYNOPSIS

/usr/atria/bin/mvfsversion [**–r**] [**–s**]

DESCRIPTION

Lists the version string of your host's MVFS, in RCS or SCCS format. This string also appears at operating system startup.

OPTIONS AND ARGUMENTS

Default: The MVFS version string is displayed in SCCS format.

–s Same as default.

–r Displays the version string in RCS format.

EXAMPLES

- Display the MVFS version string in RCS format.

```
% mvfsversion –r
$Header: MVFS Release 2.0 (Fri Apr 11 16:51:23 EST 1994) $
```

SEE ALSO

mfscache, mvfslog, mvfsstat, mvfsstorage, mvfstime

NAME pathnames_ccase – ClearCase pathname resolution, view context, and extended namespace

SYNOPSIS

- View-Extended Pathname:

  /view/*view-tag*/*full-pathname*

- VOB-Extended Pathname:

Element:	*element-pname*@@
Branch:	*element-pname*@@*branch-pname*
Version:	*element-pname*@@*version-selector*
VOB symbolic link:	*link-pname*
Derived object:	*derived-object-pname*@@*DO-ID*

DESCRIPTION

This manual page describes ClearCase's extensions to the standard file/directory namespace provided by the operating system. These extensions can be used only on a host that supports the ClearCase *multiversion file system* — the *MVFS*. Code that implements the MVFS is linked with a host's operating system, either statically (which requires generation of a new version of the operating system that includes the MVFS) or dynamically (the MVFS code is loaded at system startup time).

All ClearCase data is logically accessed at locations under *VOB-tags* (VOB mount points). Physically, some data is stored in a VOB storage pool — for example, a checked-in version of a file element. Other data is stored in a view's private storage area — for example, a checked-out version of a file element. Collectively, all such file system objects are termed *MVFS objects* — files, directories, and links.

VIEW CONTEXTS

A pathname can access ClearCase data only if it has a *view context*:

- **Set view context** — A process, typically a shell, created with the *setview* command is said to have a *set view* context. That process, along with all of its children, is "set to the view".
- **Working directory view context** — You can change the current working directory of a process to a view-extended pathname:

  ```
  % cd /view/david/vobs/proj
  ```

  Such a process is said to have a *working directory view* context. (The process may or may not also have a *set view* context.)
- **View-extended pathname** — A pathname can specify its own view context, regardless of the current set view or working directory view contexts, if any.

KINDS OF PATHNAMES

The following sections describe the kinds of pathnames you can use with ClearCase.

Standard Pathnames

A standard pathname is either full or relative:

- A *full pathname* begins with a slash character (/):

  ```
  /vobs/proj
  /usr/bin/cc
  ```

  A full pathname is interpreted in the process's set view context. An error occurs if you attempt to use a full pathname to access ClearCase data in a process that is not set to a view.

- A *relative pathname* does not begin with a slash character:

  ```
  foo.c
  ../lib
  motif/libX.a
  ```

  A relative pathname is interpreted in the process's working directory view context, if it has one. Otherwise, it uses the process's set view context. If a process has neither kind of view context, an error occurs.

A standard pathname can reference any kind of file system object: For example, */vobs/proj/BAR* references "file system object named 'BAR', as seen through the current view". This can be any of the following:

- **Version** — If "BAR" names an element, the pathname references the version of that element selected by the current view's config spec.
- **VOB symbolic link** — "BAR" can name a VOB symbolic link that is visible in the current view. Depending on the command, the link may or may not be traversed.
- **Derived object** — "BAR" can name a derived object that was built in the current view, or was winked-in to the view.
- **View-private object** — "BAR" can name a view-private object (including a checked-out version) located in the current view's private storage area.
- **Non-MVFS object** — "BAR" can name an object that is not under ClearCase control, such as objects in your home directory or in */usr/bin*.

Using standard pathnames to reference MVFS objects is termed *transparency*: a view's *view_server* process resolves (converts) the standard pathname into a reference to the appropriate MVFS object. In essence, transparency makes a VOB appear to be a standard directory tree.

ClearCase Extended Pathnames

The MVFS supports two kinds of extensions to the standard pathname scheme:

- You can add two pathname components to the beginning of any full pathname, turning it into a *view-extended pathname*:

  ```
  /view/david/vobs/proj/foo.c                    (view–extended full pathname)
  ```

In certain situations, a relative pathname can include a view specification:

```
../../david/vobs/proj/foo.c                    (view–extended relative pathname)
```

- You can add characters to the end of a relative or full pathname, turning it into a *VOB-extended pathname*. VOB-extended pathnames that specify versions of elements are the most commonly used; they are termed *version-extended pathnames*.

```
foo.c@@/main/12                                (version–extended pathname)
/vobs/proj/foo.c@@/main/motif/4                (version–extended pathname)
foo.c@@/RLS4.3                                 (version–extended pathname)
foo.c@@/main                                   (VOB–extended pathname to a branch)
foo.c@@                                        (VOB–extended pathname to an element)
hello.o@@15-Sep.08:10.439                      (VOB–extended pathname to a derived object)
```

VIEW-EXTENDED PATHNAMES

A *view-extended pathname* is a standard pathname, along with a specification of a ClearCase view. For example, */view/david/vobs/proj/BAR* references "file system object named 'BAR', as seen through view 'david'". A view-extended pathname can access any kind of file system object, as described in section "Standard Pathnames" above.

The Viewroot Directory / View-Tags

In most view-extended pathnames, a full pathname is prepended with two components: the name of the host's *viewroot* directory and the *view-tag* of a particular view. The *viewroot* directory is a virtual data structure, whose contents exist only in MVFS buffers in main memory. Each view is made accessible to standard programs and ClearCase programs through a *view-tag* entry in the viewroot directory. No standard command or program can modify this directory. Only a few ClearCase commands use or modify it: *mkview*, *mktag*, *rmtag*, *rmview*, *startview*.

The viewroot directory is activated by a standard *mount*(1M) command, which considers the virtual data structure to be a file system of type MVFS. The ClearCase-standard pathname of the viewroot directory is */view*. See the *init_ccase* and *viewroot* manual pages for details.

SYMBOLIC LINKS AND THE VIEW-EXTENDED NAMESPACE

Pathnames are resolved component-by-component by the operating system kernel and the ClearCase MVFS. When a UNIX symbolic link or VOB symbolic link is traversed, the note above applies: a full pathname needs a set view context to access ClearCase data. Thus, a symbolic link whose text is a full pathname ...

```
/vobs/aardvark -> /vobs/all_projects/aardvark
```

… will be interpreted in the current set view context. If the process has no set view context, traversing such a symbolic link will fail.

VOB-EXTENDED PATHNAMES

ClearCase's transparency feature enables you to use standard pathnames to access version-controlled data; the *view_server* does the work of locating the data. But you can also bypass transparency and "do the work yourself":

- You can access any version of an element by using its *version-ID*, which specifies its exact version-tree location:

```
sort.c@@/main/motif/4
```

- If a version has been assigned a version label, you can access it using the label:

```
sort.c@@/main/motif/RLS_1.3          (branch and version label)
sort.c@@/RLS_1.3                     (version label only)
```

  Typically, you can use *just* the label, without having to specify the branch on which the labeled version resides — see "Version Labels in Extended Namespace" below.

- You can access any element object or branch object directly:

```
sort.c@@                             (element object)
sort.c@@/main                        (branch object)
sort.c@@/main/motif                  (branch object)
```

- You can access any derived object directly, no matter what view it was created in:

```
sort.o@@13-Aug.09:45.569             (derived object created on 13–Aug)
sort.o@@23-Sep.19:09.a50f            (derived object created on 23–Sep)
```

The pathnames in the above examples are termed *VOB-extended pathnames*. A VOB's file/directory namespace is extended in two ways from the standard namespace: one extension enables direct access to elements, branches, and versions; the other enables direct access to derived objects. Both extensions allow you to access objects not visible in your own view (and, perhaps, not currently visible in any other view, either).

Extended Namespace for Elements, Branches, and Versions

An element's version tree has the same form as a standard directory tree (Figure 20):

Component of Version Tree	Component of Directory Tree in Extended Namespace
element	*root of tree*: The element itself appears to be a directory, which contains a single subdirectory, corresponding to the *main* branch. (It can also contain some version labels — see "Version Labels in Extended Namespace" below.)
branch	*subdirectory*: Each branch appears to be a directory, which contains files (individual versions), directories (subbranches), and links (version labels).
version	*leaf*: Each version appears to be a leaf of a directory tree. For a file element, the leaf contains text lines or binary data, and can be processed with standard commands like *cat*, *diff*, and *cmp*. For a directory element, the leaf contains a directory structure, and can be processed with standard commands like *ls* and *cd*.

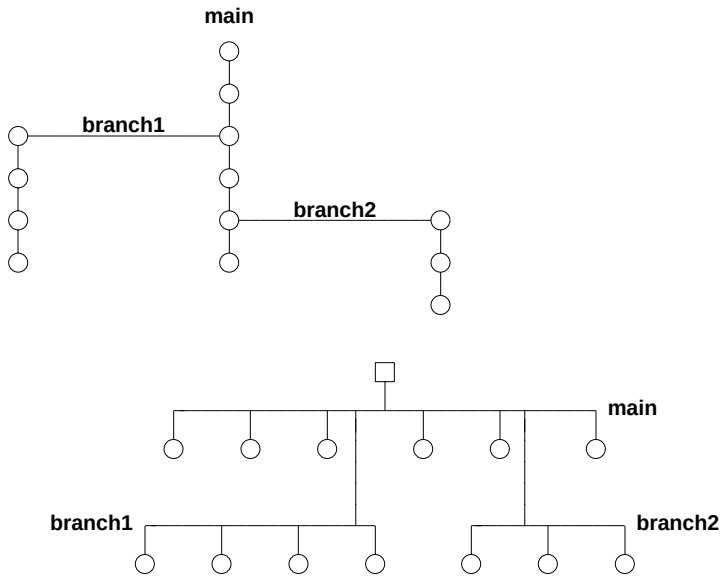


Figure 20. Version Tree and Extended Namespace

Accordingly, any location within an element's version tree can be identified by a pathname in this extended namespace:

`sort.c@@`	*(specifies an element)*
`sort.c@@/main`	*(specifies a branch)*
`sort.c@@/main/branch1`	*(specifies a branch)*
`sort.c@@/main/branch1/2`	*(specifies a version)*
`doctn/.@@/main/3`	*(special case: extra component is required element in VOB's top–level directory)*

Extended Naming Symbol

The pathname examples above all incorporate the *extended naming symbol*, `@@`. This symbol is required to effect a "switch" from the standard file/directory namespace to the extended element/branch/version namespace. There are two equivalent ways to think of `@@`:

- When appended to the name of any element, the extended naming symbol turns off transparency (automatic version-selection). Thus, you must specify one of the element's versions explicitly.
- The extended naming symbol is part of an element's "official" name. For example, *foo.c* is the name of a version (the particular version that appears in the view); *foo.c*@@ is the name of the element itself.

NOTE. The establishment of `@@` as the extended naming symbol occurs at system startup time with a file system table entry. Thus, different symbols might be used on different hosts. See the *init_ccase* manual page for details.

Version Labels in Extended Namespace

Version labels appear in the extended namespace as hard links. If version */main/4* of an element is labeled *RLS_1*, then the extended namespace "directory" corresponding to the element's *main* branch lists both *4* and *RLS_1* as hard links to the version:

```
% ls –il sort.c@@/main
  246 -r--r--r--    1 drp      user         217 Oct  6 21:12 4
  :
  :
  246 -r--r--r--    1 drp      user         217 Oct  6 21:12 RLS_1
```

If the label type was created with the "once-per-element" restriction, an additional hard link to the labeled version appears in the element's top-level directory:

```
% ls –il sort.c@@
  246 -r--r--r--    1 drp      user         217 Oct  6 21:12 RLS_1
```

In this case, all the following are equivalent extended pathnames to the labeled version:

```
sort.c@@/RLS_1                  (version label at top level of element)
sort.c@@/main/4                 (version–ID)
sort.c@@/main/RLS_1             (version label at branch level)
```

(The "once-per-element" restriction is the *mklbtype* default. A `mklbtype -pbranch` command creates a label type that can be used once on each branch of an element.)

Pathnames Involving More Than One Element

A VOB can implement an arbitrarily deep directory structure. Thus, a pathname can involve several elements — for example:

```
/vobs/proj/src/include/sort.h
```

If *proj* is the VOB's root directory element, then *src* and *include* also name directory elements, and *sort.h* names a file element.

RULE: once a pathname "crosses over" into the extended namespace with `@@`, you must specify a version for *each* succeeding element in the pathname. For example:

`/vobs/proj/src/include@@/main/4/sort.h/main/LATEST`
: Automatic version selection for elements *proj* and *src*; cross over to extended namespace at directory element *include*, specifying a version of *include* and a version of *sort.h*.

`/vobs/proj/src@@/RLS_1/include/RLS_1/sort.h/RLS_1`
: Automatic version selection for element *proj* only; cross over to extended namespace at directory element *src*, specifying the version labeled *RLS_1* of each succeeding element.

`/vobs/proj@@/main/1/src/main/4` (invalid)
`/vobs/proj/.@@/main/1/src/main/4` (valid)
: *Special case*: When crossing over into extended namespace at the VOB root directory (that is, at the VOB-tag or VOB mount point), you must use `/.@@` instead of `@@` .

The extended naming symbol need be used only once in a pathname, to indicate the crossover into extended namespace. You can, however, append it to any element name:

```
/vobs/proj/src@@/RLS_1/include@@/RLS_1/sort.h@@/RLS_1
```

Reading and Writing in the Extended Namespace

A VOB-extended pathname references an object in a VOB database. The reference can either read or write the database — that is, either query meta-data or modify meta-data:

```
% cleartool mklabel RLS2.1 util.c@@/RLS2.0      (attach an additional label to a version)
% cleartool rmattr BugNum util.c@@/main/3       (remove an attribute)
```

For a version, an extended pathname can also read the version's data, but cannot write or delete it:

```
% grep 'env' util.c@@/main/rel2_bugfix/1        (valid)
% rm util.c@@/main/rel2_bugfix/1                (invalid)
ERROR: util.c@@/main/rel2_bugfix/1 not removed: Read-only file system.
```

Extended Namespace for Derived Objects

ClearCase's extended namespace allows multiple derived objects to exist at the same standard pathname. This parallels the fact that the multiple versions of an element all exist at the same standard pathname — but the two extensions work differently. Derived objects created at the same location are distinguished by their unique derived object identifiers, or *DO-IDs*:

```
sort.o@@14-Sep.09:54.418
sort.o@@13-Sep.09:30.404
sort.o@@02-Sep.16:23.353
  ⋮
```

An extended name provides access only to the derived object's meta-data in the VOB database — principally, its configuration record. To access a DO's file system data (stored in its data container), you must use a standard pathname (*sort.o*) in some view, or a view-extended pathname (*/view/david/vobs/sort.o*).

Navigating the VOB-Extended Namespace

You can use standard directory-navigation commands — *cd*, *ls*, *pwd*, and so on — in a VOB's extended namespace. For example, these are two equivalent ways to display the contents of an old version:

- Use a version-extended pathname from a standard directory:

  ```
  % cat util.c@@/main/rel2_bugfix/1
  ```

- Go to branch "directory" in the VOB-extended namespace, then display the version:

  ```
  % cd util.c@@/main/rel2_bugfix
  % cat 1
  ```

In VOB-extended namespace, elements and branches are directories; you can change to such directories with *cd*; you can lists their contents — branches and versions — with *ls*.

You can access versions of file elements as ordinary files, with *cat*, *diff*, and so on — even executing versions that happen to be compiled programs or scripts. You can access versions of file elements as ordinary directories, with *cd*, *ls*, and so on — but not with file-oriented commands, such as *cat*.

Special View-Tag Reported by 'pwd'. When you have changed to a VOB-extended namespace directory, the *pwd*(1) command reports your current working directory as under a special view-tag: For example:

```
% cd /view/akp_vu/vobs/proj/special@@
% pwd
/view/akp_vu@@/vobs/proj/main/4/special
```

The special view-tag `akp_vu@@` appears as a separate entry from `akp_vu` in your host's viewroot directory. When in the context of a special view-tag, version-selection is suppressed completely — to access a particular version of any file or directory element, you must specify the version explicitly. ClearCase periodically deletes these special entries, on a least-recently-used basis.

Exiting from VOB-Extended Namespace. To exit VOB-extended namespace, *cd* to a standard full pathname or a view-extended pathname. (The pathname can specify a VOB or non-VOB location.) For example:

```
% cd /vobs/proj/src@@/main                    (enter VOB–extended namespace)
% pwd
/view/david@@/vobs/proj/main/4/src/main
% cd /vobs/proj                               (exit VOB–extended namespace)
% pwd
/vobs/proj
```

Repeated use of `cd ..` does not work as you might expect. You do not exit extended namespace where you entered it; instead, you ascend through all the extended-namespace directories listed by *pwd*. For example:

```
% cd util.c@@/main/rel2_bugfix
% ls
0       1       2       LATEST
% pwd
/view/drp_fix@@/usr/hw/main/1/src/main/2/util.c/main/rel2_bugfix
% cd ../../..
% pwd
/view/drp_fix@@/usr/hw/main/1/src/main/2
% cd ../..
% pwd
/view/drp_fix@@/usr/hw/main/1/src
% cd ../../../
% pwd
/view/drp_fix@@/usr/hw
```

SEE ALSO

derived_object, init_ccase, query_language, version_selector, viewroot, wildcards_ccase

NAME profile_ccase – cleartool user profile: .clearcase_profile

SYNOPSIS

command_name flag
⋮

DESCRIPTION

The *cleartool* user profile is an ordered set of rules that determine the comment option default for one or more *cleartool* commands. Many *cleartool* commands accept user comments with the `–c`, `–cq`, `–cqe`, or `–nc` option. If you specify none of these options, *cleartool* invokes one of them by default — the option invoked varies from command to command.

If *cleartool* finds a file named *.clearcase_profile* in your home directory, it checks to see if it contains a comment rule that applies to the current command. If so, it invokes the comment option indicated by that rule. No error occurs if this file does not exist; *cleartool* just invokes the command's standard comment default (discussed below).

An alternate name for the user profile can be specified with the environment variable CLEARCASE_PROFILE. Its value should be a full pathname.

HOW CLEARTOOL SELECTS A COMMENT RULE

For a given command, *cleartool* consults the user profile to determine which comment rule, if any, applies. The method is similar to the one used by the *view_server* process to evaluate a config spec:

- *cleartool* examines the first comment rule in the user profile and decides whether it applies to the command.
- If the rule does not apply, *cleartool* goes on to the next rule in the file; it repeats this step for each succeeding rule until the last.
- If *no* comment rule applies, *cleartool* invokes the standard comment default for the command.

cleartool uses the first comment rule that applies. Therefore, the *order* of rules in the user profile is significant. For example, to ensure that you are always prompted for a comment when you create a directory element, you must place a rule for the *mkdir* command before any more general rule that might also apply to *mkdir*, such as `* –nc`.

COMMENT RULE SYNTAX

Comment rules must be placed on separate lines. Extra white space (space, tab) is ignored.

Comments begin with a pound sign (#) character. For example:

```
#element rules
mkelem -cqe   #prompt for comment for each new element being created
  ⋮
```

All other lines must be have two tokens, separated by white space:

command-name flag

- *command-name* must be one of these:

  checkin **mkeltype**
  checkout **mkbrtype**
  mkdir **mklbtype**
  mkelem **mkattype**
  mkpool **mkhltype**
  mkvob **mkrptype**
  mktrtype

  … or an asterisk (*****), which matches all these names

- *flag* must be one of these: **–nc**, **–cqe**, **–cq**

If you do not provide a comment rule for one of the commands above (*checkin*, *checkout*, and so on), *cleartool* uses –cqe as its default comment option. *cleartool* uses –nc as the default for all other commands that accept comments.

EXAMPLES

- Never prompt for a comment.

```
*  -nc
```

- During a *checkin* operation, prompt for a comment for each element. During a *mkdir* operation, prompt for a single comment to be applied to all the new directories. In all other cases, do not prompt at all.

```
checkin  -cqe
mkdir    -cq
*        -nc
```

SEE ALSO

cleartool, config_spec, xclearcase

NAME promote_server – change storage location of derived object data container

SYNOPSIS

invoked by clearmake, if necessary, when it performs a wink-in

DESCRIPTION

The *promote_server* program migrates a derived object's data container file from private storage to shared storage. When *clearmake* winks–in a derived object that was previously unshared, it automatically invokes *promote_server* to copy the data container file from view-private storage to a VOB storage pool. (The original data container remains in view-private storage; use the standard *rm*(1) command to remove it.)

NOTE: *clearmake* also migrates a derived object's configuration record from private storage to shared storage at the same time. This work is performed by *clearmake* itself, not by *promote_server*

The destination storage pool is determined by the derived object's pathname. By definition, this pathname is under a VOB-tag (mount point) — that is, the derived object is "in" some VOB directory. The derived object storage pool to which the directory element is assigned is the destination of the promotion. (Some build scripts create multiple hard links, in different directories, to a derived object. In this case, the data container is promoted to the storage pool of just one of the directories.)

clearmake invokes *promote_server* by making a request to the ClearCase master server, *albd_server*. *promote_server* runs as the owner of the view in which the data container to be copied resides. This ensures that the data container is readable.

After promoting a derived object, the *promote_server* remains active for several minutes to ensure that subsequent promotions from the same view are processed with the least overhead. During this time, the *promote_server* remains associated with the view from which the DO was promoted; if two users try to promote DOs from the same view, at the same time, they share (serially) the same *promote_server*.

NOTE

Never run *promote_server* manually. It should only be invoked by *clearmake*.

SEE ALSO

albd_server, clearmake, view_server

NAME query_language – select objects by their meta-data / find, findmerge, version-selector, config spec

SYNOPSIS

Query Primitives:

query-function **(** *arg-list* **)**
attribute-type-name **==** *value*
attribute-type-name **!=** *value*
attribute-type-name **<** *value*
attribute-type-name **<=** *value*
attribute-type-name **>** *value*
attribute-type-name **>=** *value*

Compound Queries:

query **&&** *query*
query **||** *query*
! *query*
(*query* **)**

DESCRIPTION

The ClearCase *query language* is used to formulate queries on *VOB databases*. It includes logical operators similar to those in the C programming language. ClearCase uses a query to search one or more VOB database(s), and returns the names of objects: versions, branches, elements, and/or *VOB symbolic links*. A query may return a single object, many objects, or no objects at all.

Queries as Version Selectors

You can use a query in a *version selector* in the following contexts:

- in *cleartool* command-line options
- in configuration rules — see the *config_spec* manual page
- in version-extended pathnames — see the *pathnames_ccase* manual page

A query in a version selector must be enclosed in braces ({...}).

When a query is applied to a single branch, ClearCase selects the most recent version on that branch that satisfies the query. For example:

```
% cleartool describe –ver '/main/{attype(QAed)}' util.c
```

Using a query without a *branch pathname* causes an element's entire version tree to be searched. If the query returns a single version, the version-selection operation succeeds; the operation fails if the query returns no version (`not found`) or returns more than one version (`ambiguous`). For example:

```
% cleartool describe –ver '{attype(QAed)}' util.c
cleartool: Error: Ambiguous query: "{attype(QAed)}"
```

Queries in the 'find' Command

You can also use queries in the *find* command. In this context, the query need not be, but can be, enclosed in braces ({...}). The query returns the names of all matching objects. For example:

```
% cleartool find util.c –ver 'attype(QAed)' –print
util.c@@/main/1
util.c@@/main/3
```

QUERY PRIMITIVES

A query primitive evaluates to TRUE or FALSE. A TRUE value selects an object, such as an element, branch, or version; a FALSE value excludes it.

A query *must* be enclosed in quotes if it includes spaces. You may also need to enclose a query in quotes to prevent shell-level interpretation of characters such as `(` (open parenthesis). Quoting parentheses in config specs is not required.

The query language includes these primitives:

attribute-type-name comparison-operator value

... where *comparison-operator* is one of the following:

```
==   !=   <   <=   >   >=
```

Examples:

```
BugNum==4053
BugNum>=4000
Status!="tested"
```

This primitive is the only one that does not use "function" notation. It is TRUE if the object itself has an attribute of that type *and* the value comparison holds. Use the **attr_sub** primitive to test whether an object or its subobjects has a particular attribute (for example, an element or its branches and versions).

NOTE: If no attribute named *BugNum* has been attached to an object, then `!BugNum==671` is TRUE, but `BugNum!=671` is FALSE. The second query would be true if an attribute of type *BugNum* exists, but has a different value.

attr_sub (*attribute-type-name*, *comparison-operator*, *value*)

With elements: TRUE if the element or any of its branches or versions has an attribute of type *attribute-type-name* that satisfies the specified comparison with *value*.
With branches: TRUE if the branch or any of its versions has an attribute of type *attribute-type-name* that satisfies the specified comparison with *value*.
With versions: TRUE if the version itself has an attribute of type *attribute-type-name* that satisfies the specified comparison with *value*.

attype (*attribute-type-name*)

With elements: TRUE if the element itself has an attribute of type *attribute-type-name*.
With branches: TRUE if the branch itself has an attribute of type *attribute-type-name*.
With versions: TRUE if the version itself has an attribute of type *attribute-type-name*.

attype_sub (*attribute-type-name*)

With elements: TRUE if the element or any of its branches or versions has an attribute of type *attribute-type-name*.
With branches: TRUE if the branch or any of its versions has an attribute of type *attribute-type-name*.
With versions: TRUE if the version itself has an attribute of type *attribute-type-name*.

brtype (*branch-type-name*)

With elements: TRUE if the element has a branch named *branch-type-name*.
With branches: TRUE if the branch is named *branch-type-name*.
With versions: TRUE if the version is on a branch named *branch-type-name*.

created_by (*login-name*)

In all cases, TRUE if the object was created by the user *login-name* (as shown by the *describe* command).

created_since (*date-time*)

In all cases, TRUE if the object was created since *date-time*, which uses the standard date-time syntax. (See the *lshistory* manual page.

eltype (*element-type-name*)

In all cases, TRUE if the element to which the object belongs is of type *element-type-name*.

hltype (*hlink-type-name*)
hltype (*hlink-type-name* **, ->**)
hltype (*hlink-type-name* **, <-**)

In all cases, TRUE if the object is either end of a hyperlink (first form) named *hlink-type-name*, or is the "from" end of a hyperlink (second form), or is the "to" end of a hyperlink (third form).

lbtype (*label-type-name*)

In all cases, TRUE if the object itself is labeled *label-type-name* (hence, only true for versions).

lbtype_sub (*label-type-name*)

With elements: TRUE if any version of element has a version that is labeled *label-type-name*.
With branches: TRUE if any version on branch has a version that is labeled *label-type-name*.
With versions: TRUE if the version itself is labeled *label-type-name*.

merge (*from-location* **,** *to-location*)

In all cases, TRUE if the element to which the object belongs has a merge hyperlink (default name: *Merge*) connecting the *from-location* and *to-location*. You can specify either or both locations with a branch pathname or a version selector. Specifying a branch produces TRUE if the merge hyperlink involves any version on that branch. The branch pathname must be complete (for example, */main/rel2_bugfix*, not simply *rel2_bugfix*).

needs_merge (*from-branch-pname*, *to-branch-pname*)

Use of this query primitive is discouraged, and is no longer supported. Its functionality is now implemented by the *cleartool* subcommand *findmerge*. Using this primitive causes a warning message to appear. See the Release 1.x *ClearCase Reference Manual* for syntax details.

pool (*pool-name*)

In all cases, TRUE if the element to which the object belongs has a *source* or *cleartext* pool named *pool-name*.

trtype (*trigger-type-name*)

In all cases, TRUE if the element to which the object belongs has an attached or inherited trigger named *trigger-type-name*.

version (*version-selector*)

With elements: TRUE if the element has a version with the specified *version-selector*.
With branches: TRUE if the branch has a version with the specified *version-selector*.
With versions: TRUE if the version itself has the specified *version-selector*.

Note that in this context, *version-selector* cannot itself contain a query. For example, `version(REL1)` is valid, but `version(lbtype(REL1))` is not.

LOGICAL OPERATORS

Primitives can be combined into expressions with logical operators. An expression can take any of these forms, where *query* is a primitive or another expression:

- *query* || *query* (logical OR)
- *query* && *query* (logical AND)
- ! *query* (logical NOT)
- (*query*) (grouping to override precedence)

OPERATOR PRECEDENCE

The precedence and associativity of the operators for attribute comparisons and formation of logical expressions are the same as in the C programming language:

- highest precedence: ! (right associative)
- lower precedence: < <= > >= (left associative)
- lower precedence: == != (left associative)
- lower precedence: **&&** (left associative)
- lowest precedence: || (left associative)

EXAMPLES

NOTE: Examples show query language in typical contexts, like *find* and the version selector. For additional examples, see the manual pages listed in the "See Also" section.

- Display the latest version of *test.c* for which the attribute *QAed* has the value *Yes*.

```
% cat test.c@@/main/'{QAed=="Yes"}'
```

- Attach the label *REL6* to the version of *test.c* that is already labeled *REL5*.

```
% cleartool mklabel –ver '{lbtype(REL5)}' REL6 test.c
Created label "REL6" on "test.c" version "/main/4".
```

- Attach an attribute to the latest version of *test.c* created since yesterday at 1 PM by user "block".

```
% mkattr –ver '{created_since(yesterday.13:00)&&created_by(block)}' \
      QAed '"No"' test.c
Created attribute "QAed" on "test.c@@/main/5".
```

- List each branch named *rel2_bugfix* that occurs in an element to which a trigger named *mail_all* has been attached.

```
% cleartool find . –branch 'brtype(rel2_bugfix)&&trtype(mail_all)' –print
./util.c@@/main/rel2_bugfix
```

SEE ALSO

cleartool subcommands: describe, find, lshistory
config_spec, pathnames_ccase, version_selector

NAME registry_ccase – ClearCase storage registry for VOBs and views

SYNOPSIS

- Registry directory:

  /usr/adm/atria/rgy

- VOB registry:

  vob_object
  vob_tag

- View registry:

  view_object
  view_tag

- Configuration files:

  rgy_hosts.conf
  rgy_region.conf
  rgy_svr.conf
  vob_tag.sec

DESCRIPTION

Each ClearCase host in the network has a *registry directory*: subdirectory *rgy* of */usr/adm/atria*, the ClearCase administration directory. (This directory is */var/adm/atria* on some platforms.) On most hosts, the registry directory contains only the *rgy_hosts.conf* and *rgy_region.conf* configuration files. On one host in the network, the *registry server host*, the registry directory contains information on all the VOBs and views in the local area network, organized into the following files:

vob_object	Registry of VOB storage directories
view_object	Registry of view storage directories
vob_tag	Registry of VOB-tags
view_tag	Registry of view-tags

You should never need to edit the four registry files on the registry server host manually. The following *cleartool* subcommands all add, delete, or modify registry file entries:

Command	**Registry Files Affected**	**Change**
mktag -view	*view_tag*	Add or replace entry
mktag -vob	*vob_tag*	Add or replace entry
rmtag -view	*view_tag*	Delete entry
rmtag -vob	*vob_tag*	Delete entry
mkview	*view_tag*	Add entry

	view_object	Add entry
rmview	*view_object*	Delete entry
rmview -tag	*view_object*	Delete entry
	view_tag	Delete entry
mkvob	*vob_tag*	Add entry
	vob_object	Add entry
rmvob	*vob_tag*	Delete entry
	vob_object	Delete entry
register -view	*view_object*	Add or replace entry
register -vob	*vob_object*	Add or replace entry
unregister -view	*view_object*	Delete entry
unregister -vob	*vob_object*	Delete entry

The *lsview* and *lsvob* commands read and report information from the registry files.

REGISTRY SERVER

One host in the local area network is designated as the *registry server host*. The name of this host must appear in the file */usr/adm/atria/rgy/rgy_hosts.conf* on each host in the network. The ClearCase *albd_server* program running on the registry server host acts as the "registry server" process: it fields RPC requests for registry information from ClearCase client programs (and other server programs) around the network.

NETWORK REGIONS

A local area network can be conceptually partitioned into multiple ClearCase *network regions*. Each region is a consistent "naming domain": all hosts in the same region must be able to access ClearCase physical data storage (that is, all VOB storage directories and view storage directories) using the same full pathnames. For example, all hosts in a network region might access a view storage directory on host *neptune* using this *automount*(1M)-style pathname:

```
/net/neptune/shared_views/rls3_clean.vws
```

Hosts in another network region might use a different name to access the same view storage directory:

```
/net/neptune_gw/shared_views/rls3_clean.vws
```

These hosts might be on a subnet that uses a different network interface to host *neptune*. Hosts that use a nonstandard auto-mount program, or that don't use an auto-mount program at all, may also need to be placed in separate network regions.

There is no formal mechanism for defining a network region. Each host exists in exactly one network region, and that region is named in the file */usr/adm/atria/rgy/rgy_hosts.conf*.

Conceptually, each network region has its own *view-tags registry* and *VOB-tags registry*. However, the registry server host stores the only copies of the view-tag and VOB-tag files; each view-tag and VOB-tag entry includes a `-region` field, which assigns the tag to a particular region.

FORMAT OF REGISTRY FILES

The following sections describe the fields in the various registry files.

vob_object

Each VOB storage directory in the network has one entry in the *vob_object* file. The entry is a single text line with these fields:

–entry	**vob_object**
–hostname	The host on which the VOB storage directory resides.
–local_path	A standard full pathname to the VOB storage directory on that host.
–vob_replica	The unique identifier (UUID) of the VOB. (For a replicated VOB, maintained by the Atria MultiSite product, this *VOB replica UUID* identifies the particular VOB replica at your site.)
–vob_family	The *VOB family UUID*, which is shared by all replicas of the same VOB. Replicas are created and maintained by the Atria MultiSite product. Even if the VOB is not replicated, this UUID is different from the VOB replica UUID.

vob_tag

Each VOB storage directory in the network can have one VOB-tag per network region, and each VOB-tag has an entry in the *vob_tag* file. The entry is a single text line with these fields:

–entry	**vob_tag**
–tag	The VOB-tag, which is a full pathname. A VOB's tag is the same as its mount point.
–global_path	A standard full pathname to the VOB storage directory that is valid on all hosts within the network region.
–hostname	The host on which the VOB storage directory resides.
–mount_access	A keyword: "private" or "public".
–mount_options	A system-dependent character string, which records options to be invoked when the VOB is mounted on a host.
–region	The network region.
–vob_replica	Same as the like-named field in the *vob_object* file (see above).
–title	(optional) The tag comment supplied with a `-tcomment` option on *mkvob* or *mktag*.

view_object

Each view storage directory in the network can have one entry per network region in the *view_object* file. The entry is a single text line with these fields:

–entry	**view_object**
–hostname	The host on which the view storage directory resides.

–local_path	A standard full pathname to the view storage directory on that host.
–view_uuid	The view's unique identifier (UUID).

view_tag

Each view storage directory in the network can have one entry per network region in the *view_tag* file. The entry is a single text line with these fields:

–entry	**view_tag**
–tag	The view-tag, which takes the form of a simple file name. When a view is active on a host, its view-tag appears as an entry in the host's *viewroot* directory (*/view*).
–hostname	The host on which the view storage directory resides.
–global_path	For all machines in this network region, the full pathname of the view storage directory.
–region	The network region.
–view_uuid	The view's unique identifier (UUID).
–title	(optional) The tag comment supplied with a `-tcomment` option on *mkview* or *mktag*.

rgy_hosts.conf

A one-per-line list of the hosts that run *registry_server* processes. This file must exist on every ClearCase host. In the current implementation, *rgy_hosts.conf* names the single registry server host for the network. The region name can be any character string that does not include white space.

rgy_svr.conf

On each host named in *rgy_hosts.conf*, the *rgy_svr.conf* file includes a single keyword, *master* or *slave*, specifying what kind of *registry_server* process runs on the host. If *rgy_hosts.conf* names a single host, that host's *rgy_svr.conf* file should contain the keyword *master*. The *rgy_svr.conf* file need not exist (its contents are ignored) on hosts not named in *rgy_hosts.conf*.

rgy_region.conf

A single word specifying the host's network region. Each ClearCase host belongs to a single network region. All hosts in a network region access VOB and view storage directories using the same "global" (network) pathnames.

vob_tag.sec

The network's *VOB-tag password*, an encrypted character string. This password is required from any user who attempts to create a public VOB-tag with *mkvob* or *mktag*. This file exists only on the registry server host. If it is empty or nonexistent, the *root* user or ClearCase administrator must place a password in it using the *rgy_passwd* utility. For example:

```
% rlogin ccsvr05 -l root                    (log in to registry server host)
Password: <enter root password>
# /usr/atria/etc/rgy_password               (invoke encrypted–password utility)
Password: <enter VOB-tag password>
```

FILES

/usr/adm/atria/rgy/vob_object
/usr/adm/atria/rgy/view_object
/usr/adm/atria/rgy/vob_tag
/usr/adm/atria/rgy/view_tag
/usr/adm/atria/rgy/rgy_hosts.conf
/usr/adm/atria/rgy/rgy_region.conf
/usr/adm/atria/rgy/rgy_svr.conf
/usr/adm/atria/rgy/vob_tag.sec

SEE ALSO

cleartool subcommands: mkview, rmview, mkvob, rmvob, mktag, lsview, lsvob, register, unregister, mount, umount
rgy_passwd

NAME rgy_passwd – create or change encrypted VOB-tag registry password

SYNOPSIS

/usr/atria/etc/rgy_passwd [**−pas·sword** *tag-registry-password*]

DESCRIPTION

Places an encrypted password in the ClearCase *VOB-tag password* file: */usr/adm/atria/rgy/vob_tag.sec* on the network's *registry server* host. This file need not already exist.

Knowledge of this password enables an administrator to create *public* VOBs. Such VOBs can be mounted by nonprivileged users, using the command `cleartool mount`. See the *mkvob*, *mktag*, and *mount* manual pages for more on public VOBs.

Security Restrictions

A high level of security is implemented through the following restrictions:

- You must execute this command on the registry server host.
- If the *vob_tag.sec* file already exists, you must be the owner of that file.
- The registry directory, */usr/adm/atria/rgy*, is protected so that only the *root* user can create the *vob_tag.sec* file.
- *rgy_passwd* maintains the access mode of the *vob_tag.sec* file at "400". (You need not use *chmod*(1) before or after entering this command.)

OPTIONS AND ARGUMENTS

By default, *rgy_passwd* prompts you to type the (new) password.

−pas·sword *tag-registry-password*
Specifies the password on the command line.

CAUTION: This is a potential security breach, since the password will remain visible in your transcript pad.

FILES

/usr/adm/atria/rgy/vob_tag.sec

DIAGNOSTICS

`Not run on registry server host`
This command must be executed on the network's registry server host.

`No permission to update file`
A *vob_tag.sec* file already exists, and you are not its owner.

SEE ALSO

cleartool subcommands: mktag, mkvob, mount
registry_ccase

NAME schemes – X Window System resources for ClearCase graphical interface

SYNOPSIS

automatically read by GUI utilities

DESCRIPTION

The ClearCase GUI utilities use *schemes*, collections of X Window System resource settings, to control their geometry, colors, and fonts. Each scheme is implemented as a separate directory. For example, the ClearCase-supplied scheme *Turner* consists of four files:

Turner/palette Defines mnemonic names for colors and fonts, using a subset of standard *cpp*(1) syntax:

```
#ifndef GAMMA_1_0
#define  TextForeground         #ffffff
#define  BasicBackground        #002e5c
#define  ScrolledListBackground #623463
  :
  :
```

Turner/Turner Specifies resources for use by the X Toolkit widgets that make up the GUI panels. These resources can be specified absolutely, or in terms of the mnemonic names defined in the *palette* file:

```
*XmText*marginHeight:  4
  :
  :
*foreground:           TextForeground
*background:           BasicBackground
```

Turner/ClearCasepalette Extends and/or overrides the "standard" *palette* definitions.

Turner/ClearCase Extends and/or overrides the "standard" *Turner* file definitions.

The two mnemonic map declaration files are combined, as are the two resource definition files. To add your own definitions, or to replace existing ones, either (1) edit one or both of the ClearCase-specific files, or (2) create your own scheme files and add them to the scheme file search path as described below in the section "Search Path for Schemes".

Note that the *palette* and *ClearCasePalette* files are not actually processed by *cpp* — they are processed by the GUI utility itself. The resources (*Turner* and *ClearCase*) apply only to the program that reads them. They are not added to the RESOURCE_MANAGER property of the root window and, therefore, do not affect other X applications.

Schemes are configured at two levels:

- Your display's X resources enable scheme usage and specify the name of a particular scheme.
- A search path capability supports maintenance of system-wide and personal schemes.

Resources for Schemes

The `scheme` resource specifies a scheme to be used by one or more GUI utilities (*xclearcase*, *xcleardiff*, *xlsvtree*). For example:

```
*scheme:  Turner                    (specifies scheme for all GUI utilities)
xclearcase*scheme:  Turner          (specifies scheme for xclearcase)
```

If you enable scheme usage but do not specify a particular scheme, the black-and-white scheme *Willis* is used. If you do not explicitly enable schemes, the Motif default resources are used when you start a GUI utility, or you may inherit a scheme automatically as described in the following section.

Monochrome and Greyscale Schemes. A user working on a monochrome monitor gets the *Willis* scheme automatically. A user working on a greyscale monitor gets the *Print* scheme automatically. You can override these assignments with the resources `*monoscheme` and `*greyscheme`, respectively. If you specify an alternative scheme, it must be located in the scheme search path, which is described in the following section.

Search Path for Schemes

The GUI utilities use a search path to find scheme directories. Effectively, the default search path is:

```
/usr/lib/X11/Schemes:/usr/atria/config/ui/Schemes
```

You can use the environment variable SCHEMESEARCHPATH to specify a colon-separated list of directories to be searched instead. Each entry on this list must be in the following standard X Toolkit form:

*pathname***/%T/%N%S**

The GUI utilities always make these substitutions:

`%T` → `Schemes`
`%N` → *scheme-name*
`%S` → (null)

For example, if your SCHEMESEARCHPATH value is:

```
/netwide/config/ui/%T/%N%S:/home/gomez/%T/%N%S
```

… and your *.Xdefaults* file includes this line:

```
*scheme:        Rembrandt
```

… then *xclearcase* reads resource schemes from these two directories:

```
/netwide/config/ui/Schemes/Rembrandt
/home/gomez/Schemes/Rembrandt
```

If the same resource is specified in two or more schemes on the search path, the last specification wins.

International Language Support. If your site uses the language resource `*xnlLanguage` to implement pathname substitutions based on national language and/or codeset, you may wish to expand customized SCHEMESEARCHPATH entries to use one or more of these optional substitution parameters:

`%L` → value of `*xnlLanguage` (*language*[_*territory*][`.`*codeset*])
`%l` → *language*
`%t` → *territory* (if any)
`%c` → *codeset* (if any)

See X Windows System Toolkit documentation for more details on constructing directory trees to store language-dependent application text files.

IRIX-5 Support. On IRIX-5 systems, ClearCase-supplied scheme files are automatically copied to */usr/lib/X11/schemes*. If you are developing a scheme for use on IRIX-5 systems, your custom scheme directory must reside in a directory named *schemes* (not *Schemes*), and the custom scheme directory and its files must mimic the file names and directory structure in a predefined scheme like */usr/lib/X11/schemes/Print*. On IRIX-5, the default scheme search path is */usr/lib/X11/schemes*, and in SCHEMESEARCHPATH entries, the `%T` is replaced by `schemes`, not `Schemes`.

If you are developing a scheme for use on both IRIX-5 and non-IRIX-5 systems, you must create two scheme dirs: *.../schemes/MyScheme*, for IRIX-5 systems, and *.../Schemes/MyScheme*, for all other systems. The IRIX-5 version of the custom scheme directory should follow the file name and directory structure in a pre-defined IRIX-5 scheme directory like */usr/lib/X11/schemes/Print*. The structure and contents of the second version of the scheme directory should be based on a predefined scheme directory like */usr/atria/config/ui/Schemes/Print*.

FILES

/usr/atria/config/ui/Schemes/Gainsborough/*
/usr/atria/config/ui/Schemes/Lascaux/*
/usr/atria/config/ui/Schemes/Leonardo/*
/usr/atria/config/ui/Schemes/Monet/*
/usr/atria/config/ui/Schemes/Print/*
/usr/atria/config/ui/Schemes/Rembrandt/*
/usr/atria/config/ui/Schemes/Sargent/*
/usr/atria/config/ui/Schemes/Titian/*
/usr/atria/config/ui/Schemes/Turner/*
/usr/atria/config/ui/Schemes/VanGogh/*
/usr/atria/config/ui/Schemes/Whistler/*
/usr/atria/config/ui/Schemes/Willis/*

SEE ALSO

xclearcase, xcleardiff, xlsvtree, clearprompt
X Toolkit documentation

NAME scrubber – remove data containers from VOB storage pools and remove DOs from VOB database

SYNOPSIS

/usr/atria/etc/scrubber [**–e** | **–f** | **–o**] [**–p** *pool*[,...] | **–k** *kind*[,...]]
[**–a** | *vob-storage-dir-pname* ...]

DESCRIPTION

The *scrubber* program deletes (*scrubs*) *data container* files from the cleartext storage pools and derived object storage pools of one or more VOBs. It also deletes corresponding derived objects from a VOB database. Only cleartext pools and derived object pools are affected; scrubbing is not defined for source pools.

SCRUBBING ALGORITHMS

scrubber implements several scrubbing algorithms, described in the following sections.

Heuristic Scrubbing

By default or with the `–o` option, *scrubber* uses a free-space-analysis heuristic: it compares a disk partition's current free-space level with a lower limit that it computed during its preceding execution (stored in file */usr/adm/atria/cache/scrubber_fs_info*):

- If the free-space level is still above the computed limit, *scrubber* performs no scrubbing at all in that partition, regardless of the state of the storage pools within it. This performance optimization allows a "quick check" to take place frequently (for example, once an hour), without much system overhead.
- If the free-space level has fallen below the limit, *scrubber* performs parameter-driven scrubbing of each storage pool in the partition.

Parameter-Driven Scrubbing

With the `–f` option, *scrubber* removes data container files from a storage pool according to the pool's *scrubbing parameter* settings. (As described above, the heuristic scrubbing algorithm can also "fall through" to this algorithm.)

When a derived object pool or cleartext pool is created with *mkvob* or *mkpool*, its scrubbing parameters are set to user-specified or default values:

maximum size	maximum pool size (specified in Kb; default=0)
reclaim size	size to which *scrubber* should attempt to reduce the pool (specified in Kb; default=0)
age	threshold to prevent premature scrubbing of recently-referenced objects (specified in hours; default=96)

Parameter-driven scrubbing proceeds as follows:

- Files are removed from a pool only if its current size exceeds its *maximum size* setting. In this case, *scrubber* begins deleting data containers that have not been referenced within *age* hours, proceeding on a least-recently-referenced basis:
- The data container for a derived object is deleted only if the DO's reference count is zero. In this case, the derived object in the VOB database is deleted, too. The associated configuration record is also deleted if no other derived object is associated with it.

- Cleartext data containers do not have reference counts; they are deleted solely on the basis of recent usage.
- Scrubbing stops when the pool's size falls below its *reclaim size* setting. But in no case does *scrubber* delete any object that has been referenced within the last *age* hours.

A maximum size of zero is a special case: this instructs *scrubber* to delete *all* data containers that have not been referenced within *age* hours, regardless of the *reclaim size* setting.

Everything-Goes Scrubbing

With the `-e` option, *scrubber* ignores a pool's scrubbing parameters, and:

- deletes all files from each cleartext pool
- deletes all files with zero reference counts from each derived object pool

To avoid deleting files that are currently being used, it does not delete any file that has been accessed in the preceding two minutes.

AUTOMATIC SCRUBBING

By default, *scrubber* is run periodically by a *crontab*(1) script. (See the *crontab_ccase* manual page). The script uses the `-f` option, so that each pool is examined individually. You can edit this script to change the scrubbing options; use *crontab* (as the *root* user) to change the scrubbing frequency.

You can scrub one or more pools "manually" at any time.

OPTIONS AND ARGUMENTS

Specifying the Scrubbing Algorithm. *Default:* Invokes the free-space-analysis heuristic described above, instead of examining pools individually.

-f Examines all specified pools individually, using the parameter-driven algorithm. This does *not* guarantee that any objects will actually be removed from the pool(s)

-e Examines all specified pools individually (as with `-f`), using the "everything-goes" algorithm.

-o Same as default.

Specifying the Pools. *Default:* All of a VOB's cleartext and derived object pools are scrubbed.

-p *pool*[,...]
Restricts scrubbing to pools with the specified name(s), which might occur in multiple VOBs. The list of pool names must be comma-separated, with no white space.

-k *kind*[,...]
Restricts scrubbing to pools of the specified kind(s). Valid kinds are `do` and `cltxt`. The list of kinds must be comma-separated, with no white space.

Specifying the VOBs. *Default:* None.

-a Scrubs all VOBs listed in the ClearCase *storage registry* whose storage directories reside on the local host. An error occurs if a VOB is listed in the registry, but cannot be found on the local host.

vob-storage-dir-pname …

One or more pathnames of VOB storage directories, indicating the particular VOB(s) to be scrubbed.

SCRUBBER LOG FILE

scrubber documents its work in the host's scrubber log file, */usr/adm/atria/log/scrubber_log*. For example, this partial report describes the results of scrubbing a derived object pool:

```
04/27/93 08:03:00 Stats for VOB betelgeuse:/usr1/vobstorage/orange.vbs
Pool ddft:

04/27/93 08:03:00  Get cntr tm 918.928979
04/27/93 08:03:00  Setup tm    10631.121127
04/27/93 08:03:00  Scrub tm    1207.099240
04/27/93 08:03:00  Total tm    12757.149346
04/27/93 08:03:00  Start size   404789  Deleted       3921  Limit size    0
04/27/93 08:03:00  Start files   20349  Deleted        121  Subdir  dels   0
04/27/93 08:03:00  Statistics for scrub of DO Pool ddft:
04/27/93 08:03:00  DO's           3671  Scrubs         121  Strands    1760
04/27/93 08:03:00  Lost refs      1790  No DO's       20228
```

The first six lines, which contain elapsed times and file statistics, are included in the report for every pool. The last three lines are specific to DO pools.

Get cntr tm	Elapsed time for first scrubbing phase: walk the file system tree to get pathname, size, and referenced-time information for each container in the pool.
Setup tm	Elapsed time for second scrubbing phase: perform setup processing specific to the kind of storage pool. For a cleartext pool, no setup is required. For a DO pool, setup is complicated; see "Processing of Derived Object Pools" below.
Scrub tm	Elapsed time for third scrubbing phase: determine which containers to delete, and then delete them.
Start size	Total size (Kb) of all the container files in the storage pool directory before this scrubbing.
Deleted	Amount of storage (Kb) reclaimed by this scrubbing.
Limit size	Desired size of the pool (Kb), as specified by the pool's *maximum size* parameter.
Start files	Total number of container files in the storage pool directory before this scrubbing.
Deleted	Number of container files deleted by this scrubbing.
Subdir dels	Number of empty subdirectories of the storage pool directory deleted by this scrubbing.
DO's	Total number of zero-reference-count DOs in the VOB database before scrubbing.
Scrubs	Total number of shared zero-reference-count DOs deleted by this scrubbing. (This should always equal the "Deleted" count above.)

Strands	Total number of *stranded DOs* deleted by this scrubbing. (These are described below.)
Lost refs	Total number of *lost DO reference counts* deleted by this scrubbing. (These are described below.)
No DO's	Total number of containers in the DO pool before scrubbing that are not associated with a zero-reference-count shared DO. (Each of these is presumably associated with a DO that is still referenced by some view, and hence cannot be scrubbed).

Processing of Derived Object Pools

For a DO pool, *scrubber* does more than simply delete old, unreferenced data containers; it also:

1. Deletes the derived object in the VOB database corresponding to the data container, and possibly its associated configuration record, as well. This occurs during the third phase of scrubbing.
2. Finds and deletes all *stranded DOs* from the VOB database: DOs that were never shared, and whose data containers have been deleted from view-private storage. (The VOB database is not updated when the DO's data file is removed or overwritten in the view, due to implementation restrictions.) There are no data containers in the DO storage pool for such DOs, since they were never shared. This occurs during the second phase of scrubbing.
3. Finds and deletes all *lost DO reference counts* from the VOB database. Such entries are an implementation artifact; they correspond to files that were created during a build, but deleted before the build completed. This occurs during the second phase of scrubbing.
4. Finds and deletes all *stranded configuration records*: CRs that do not correspond to any existing derived object.

Derived Statistics

Some interesting results can be derived from these statistics:

- Number of zero-reference-count DOs in this pool before scrubbing:

  ```
  DO's - Strands - Lost refs
  ```
  or
  ```
  Start files - No DO's
  ```

- Total number of derived object data containers in this pool after scrubbing:

  ```
  Start files - scrubs
  ```

- Total number of unreferenced data containers in this pool after scrubbing:

  ```
  Start files - scrubs - No DO's
  ```

- Total size (Kb) of the storage pool after scrubbing:

  ```
  Start size - deleted
  ```

EXAMPLES

- Force scrubbing of all mounted VOBs with a storage directory on the local host (default scrubbing performed by ClearCase *crontab* script */usr/atria/config/cron/scrubber_day.sh*).

  ```
  % /usr/atria/etc/scrubber –f –a
  ```

- Scrub cleartext pools in the VOB whose storage directory is */usr/vobstore/project.vobs*, using the free-space analysis heuristic.

  ```
  % /usr/atria/etc/scrubber –o –k cltxt /usr/vobstore/project.vobs
  ```

- Force scrubbing of the default derived object pool (*ddft*) and the pool named *do_staged* in all mounted VOBs with a storage directory on the local host.

  ```
  % /usr/atria/etc/scrubber –f –p ddft,do_staged –a
  ```

SEE ALSO

cleartool subcommands: lsdo, chpool, mkpool, rmdo
crontab_ccase, filesys_ccase, promote_server, crontab(1)

NAME softbench_ccase – ClearCase Encapsulation for SoftBench

SYNOPSIS

invoked as needed by SoftBench Broadcast Message Server

DESCRIPTION

The ClearCase Encapsulation for SoftBench enables integration of ClearCase with all of the SoftBench tools. ClearCase services and broadcasts all the messages prescribed for CM systems in the document "CASE Communique: Configuration Management Operation Specifications" from the "historical" standard.

ClearCase adds a menu to the SoftBench Development Manager, providing users with a familiar interface to ClearCase's most important version control and configuration management functions. Users can customize the SoftBench environment to add items to this menu, accessing more sophisticated features.

Users can configure the SoftBench Builder to use the ClearCase build tool, *clearmake*. All other SoftBench tools (debugger, browser, static analyzer, and so on) work within ClearCase environments by using ClearCase's transparent file access capability.

ClearCase can broadcast SoftBench messages whenever ClearCase performs a CM operation, no matter how that operation was requested: from the SoftBench or ClearCase graphical user interface, from the ClearCase command line interface, from the ClearCase API, from other SoftBench tools, and so on. This flexibility accommodates a variety of working styles without sacrificing tool integration.

SoftBench tools communicate with ClearCase through the SoftBench *Broadcast Message Server* (BMS), and two ClearCase server processes:

- *clearencap_sb* — the ClearCase *encapsulator* for SoftBench
- *sb_nf_server* — the ClearCase *notice forwarder* for SoftBench

After SoftBench has been configured to work with ClearCase, certain SoftBench commands automatically invoke ClearCase operations. When a SoftBench tool makes a configuration management request, such as VERSION-CHECK-OUT, the BMS receives the message and passes it on to the ClearCase encapsulator. (The BMS starts the encapsulator process if not already running.) The encapsulator evaluates the message and invokes the appropriate ClearCase tool, such as `cleartool checkout`.

- If the operation succeeds, the ClearCase tool sends a success message to the notice forwarder process (starting it if necessary). The notice forwarder informs the BMS that the operation succeeded.
- If the operation fails (that is, the ClearCase tool exits with a non-zero exit status), the encapsulator returns a failure message to the BMS.

In both cases, the BMS passes the final status message back to the SoftBench tool.

You can have ClearCase tools send the success messages described above, even if the operation was not initiated by a SoftBench tool:

- Make sure that the ClearCase tool and the BMS both have the environment variable DISPLAY set to the same value.

- Run the ClearCase tool in an environment with CLEARCASE_MSG_PROTO set to `SoftBench`.

An error occurs in a ClearCase tool that has CLEARCASE_MSG_PROTO set correctly, but not DISPLAY.

NOTE: HP VUE users must add the *$ATRIAHOME/bin* directory to their search path by adding a line like the following to the file */usr/lib/X11/vue/Vuelogin/Xconfig*:

```
Vuelogin*userPath: /usr/bin/X11:/bin:/usr/bin:/etc:/usr/contrib/bin:/usr/atria/bin:/usr/lib:/usr/lib/acct
```

Otherwise, the encapsulator will be unable to find ClearCase utilities.

ENCAPSULATOR TRANSCRIPT PAD

Text output produced by encapsulator operations can be placed in a file ("results_file" in the pseudo-syntax summaries in the next section). If a result file is not specified, output is directed to the encapsulator's dedicated transcript pad. The pad is created and appears on-screen the first time output is directed to it. The transcript pad window has a single menu, with these choices:

clear pad	Removes the current contents of the pad.
cancel	Interrupts the current encapsulator operation.
quit	Removes the transcript pad window from the screen. The transcript pad process continues to run and to collect output text. The window will reappear on the next operation that sends text to the pad, with the new output appended to the existing contents of the pad.

ENCAPSULATION SUMMARY

The *clearencap_sb* program handles the SoftBench messages listed in the pseudo-code syntax summary below. These conventions apply:

- The *context* parameter is replaced by the pathname(s) currently selected in the SoftBench tool.
- Virtually all other parameters are optional. A default action is taken if no value is supplied for a given parameter, or if it has the string value `"-"` (except with comments — see below).
- Many messages take optional comments. If a comment is not supplied, *clearencap_sb* prompts the user for a comment before acting on the message.

  NOTE: The comment string `"-"` does not indicate a default action; it is a one-character comment.
- Braces ({ … }) indicate that a non-default value from the message is substituted at that location.
- **DEFAULT** indicates that the user either did not supply the parameter or specified the string `"-"`.

Standard Messages

The following messages are specified in the "historical standard".

VERSION-CHECK-IN *context rev options keyword comment*

```
if (keyword == "CO-LOCK")
    cleartool checkin -c comment options context
    cleartool checkout -nc context
else if (keyword == "CANCEL")
    cleartool uncheckout { options | -keep } context
else
    cleartool checkin -c comment options context
```

VERSION-CHECK-OUT *context rev options keyword comment*

```
if (keyword == "CO")
    if (context{@@rev} not in current view)
        fail
    else
        succeed
else
    cleartool checkout -c comment options context{@@rev}
```

VERSION-COMPARE-REVS *context rev1 rev2 results_file*

```
if (results_file != DEFAULT && results_file != "*")
    if (rev1 == "-pred")
        cleartool diff -pred context{@@rev2} > results_file
    else
        cleartool diff context{@@rev1} context{@@rev2} > results_file
else
    if (rev1 == "-pred")
        cleartool xdiff -pred context{@@rev2}
    else
        cleartool xdiff context{@@rev1} context{@@rev2}
```

VERSION-INITIALIZE *context options comment*

```
cleartool mkelem -c comment options context
```

VERSION-LIST-DIR *context results_file keyword options*

```
if (keyword == "RECURSIVE")
    cleartool ls -r options context { > results_file }
else
    cleartool ls options context { > results_file }
```

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

VERSION-SET-MASTER *context configuration options*

```
if (configuration == DEFAULT)
    cleartool setcs -default options
else if (configuration == "*")
    cleartool edcs
else
    cleartool setcs options configuration
```

VERSION-SHOW-HISTORY *context results_file options*

```
cleartool lshistory options context { > results_file }
```

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

VERSION-UPDATE-DIR *context keyword options*

(no action needed with ClearCase — always succeeds)

Non-Standard Messages

The following messages are ClearCase extensions, not specified in the "historical standard".

VERSION-MAKE-DIR *context keyword options comment*
if (*keyword* == "QUERY")
prompt for *directory-name*
cleartool mkdir -c *comment options context*[*/directory-name*]

VERSION-MAKE-BRANCH *context branch-type-name rev options comment*
cleartool mkbranch {-version *rev*} -c *comment options branch-type-name context*

DERIVED-CAT-CONFIG-REC *context do-extension results_file options*
cleartool catcr *options context*{@@*do-extension*} { > *results_file* }

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

DERIVED-DIFF-CONFIG-REC *context do-extension1 do-extension2 results_file options*
cleartool diffcr *options context*{@@*do-extension1*}
context{@@*do-extension2*} { > *results_file* }

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

VERSION-MAKE-ATTRIBUTE *context options attribute-type attribute-value comment*
if (options include "-default")
cleartool mkattr -c *comment options* -default *attribute-type context*
else
cleartool mkattr -c *comment options attribute-type attribute-value context*

VERSION-GET-ATTRIBUTE *context options attribute-type results_file*
cleartool describe -short *options* -aattr *attribute-type context* { > *results_file* }

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

VERSION-MAKE-LABEL *context options label-type comment*
cleartool mklabel -c *comment options label-type context*

START-VIEW *context view_tag*
cleartool startview *view_tag*

VERSION-DESCRIBE *context options results_file*
cleartool describe *options context* { > *results_file* }

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

VERSION-LIST-CHECKOUTS *context options results_file*
cleartool lscheckout *options context* { > *results_file* }

NOTE: if *results_file* is DEFAULT, output is sent to the transcript pad.

VERSION-SHOW-VTREE *context options results_file*
if (*results_file* = DEFAULT)
cleartool xlsvtree *options context*
else
cleartool lsvtree *options context* > *results_file*

VERSION-COMPARE-FILES *context file2 result-file*

if (*result-file* != DEFAULT && result-file != "*")
    cleartool diff *context file2* > *result-file*
else
    cleartool xdiff *context file2*

If *file2* is not supplied as part of the message, or is either - or *, then *clearencap_sb* prompts the user for a file name (using the Motif file-selection dialog box).

VERSION-MERGE-REVS *context options rev*

cleartool xmerge *options* –to *context* –version *rev*

DO-COMMAND *context keyword command*

if (*command* includes the string "<context>")
    first substitute *context* for this string,
    then execute the resulting command
else
    cleartool *command context*

NOTE: If the keyword is DISPLAY, output is sent to the transcript pad.

FILES

/usr/adm/atria/log/ti_server_log error log for notice forwarder

SEE ALSO

"Using the ClearCase/SoftBench Integration" in the *ClearCase User´s Manual*.

NAME tooltalk_ccase – ClearCase Encapsulation for ToolTalk

SYNOPSIS

invoked as needed by ToolTalk Session Server

DESCRIPTION

ToolTalk™ tools communicate with ClearCase through the ToolTalk Session Server, *ttsession*, and two ClearCase server processes:

- *clearencap_tt* — the ClearCase *encapsulator* for ToolTalk
- *tt_nf_server* — the ClearCase *notice forwarder* for ToolTalk

After ToolTalk has been configured to work with ClearCase, certain ToolTalk commands automatically invoke ClearCase operations. When a ToolTalk tool makes a configuration management request, such as CM-Checkout-File, the Session Server receives the message and passes it on to the ClearCase encapsulator. (The Session Server starts the encapsulator process if not already running.) The encapsulator evaluates the message and invokes the appropriate ClearCase tool, such as `cleartool checkout`.

- If the operation succeeds, the ClearCase tool returns a success exit status to *clearencap_tt*, which sends a success reply back to the Session Server.
- If the operation fails (non-zero exit status), the encapsulator returns a failure status to the Session Server.

In both cases, the Session Server passes the final status message back to the ToolTalk tool.

A ClearCase tool can send a success message even if the operation was not initiated by a ToolTalk tool:

- Make sure that the ClearCase tool and the Session Server both have the environment variable DISPLAY set to the same value.
- Run the ClearCase tool in an environment with CLEARCASE_MSG_PROTO set to `ToolTalk`.

(An error occurs in a ClearCase tool that has CLEARCASE_MSG_PROTO set correctly, but not DISPLAY.) In this environment, the Notice Forwarder generates a success message on each applicable ClearCase operation that succeeds.

ENCAPSULATOR TRANSCRIPT PAD

Text output produced by encapsulator operations can be placed in a file ("results_file" in the pseudo-syntax summaries in the next section). If a result file is not specified, output is directed to the encapsulator's dedicated transcript pad. The pad is created and appears on-screen the first time output is directed to it. The transcript pad window has a single menu, with these choices:

clear pad	Removes the current contents of the pad.
cancel	Interrupts the current encapsulator operation.
quit	Removes the transcript pad window from the screen. The transcript pad process continues to run and to collect output text. The window will reappear on the next operation that sends text to the pad, with the new output appended to the existing contents of the pad.

ENCAPSULATION SUMMARY

The *clearencap_tt* program handles the ToolTalk RPC signatures listed here. Also listed are the corresponding ClearCase command(s) that *clearencap_tt* invokes.

CM-Checkin-File(in string *filename***)**
cleartool checkin -nc *filename*

CM-Checkout-File(in string *filename***)**
cleartool checkout -nc *filename*

CM-Revert-File(in string *filename***)**
cleartool uncheckout -keep *filename*

CM-Create-File(in string *filename***)**
cleartool mkelem -nc *filename*

CM-Create-Directory(in string *filename***)**
cleartool mkdir -nc *filename*

CM-Create-Branch(in string *filename*, **in string** *branchtype***)**
cleartool mkbranch -nc *branchtype filename*

CM-List-Items(in string *dirname* [**, in string** *results_file*] **)**
cleartool ls *dirname* > *results_file*
… or
cleartool ls *dirname* [output to transcript pad]

CM-List-Changes(in string *dirname* [**, in string** *results_file*] **)**
cleartool lshistory *dirname* > *results_file*
… or
cleartool lshistory *dirname* [output to transcript pad]

CM-List-Checkouts(in string *dirname* [**, in string** *results_file*] **)**
cleartool lscheckout *dirname* > *results_file*
… or
cleartool lscheckout *dirname* [output to transcript pad]

CM-Compare-Revs(in string *rev1*, **in string** *rev2***)**
cleartool xdiff *rev1 rev2*

CM-Set-Config(in string *filename***)**
cleartool setcs *filename*

CM-Do-Command(in string *cmdbuf***)**
execute command in *cmdbuf* [output to transcript pad]

BUILD-Build(in string *current_wd*, **in string** *target*, **in string** *results_file***)**
cd *current_wd* ; start terminal emulator; clearmake *target* > *results_file*

BUILD-Do-Command(in string *current_wd*, **in string** *cmdbuf***)**
cd *current_wd* ; execute command in *cmdbuf* [output to transcript pad]

FILES

/usr/atria/config/tooltalk/*	files for ToolTalk Type Compiler
/usr/adm/atria/log/ti_server_log	error log for notice forwarder

SEE ALSO

"Using the ClearCase/ToolTalk Integration" in the *ClearCase User´s Manual*.

NAME type_manager – programs for managing contents of element versions

SYNOPSIS

- Type manager directory:

  /usr/atria/lib/mgrs/*manager-name*

- Methods, some or all of which are supported by each type manager:

  annotate, **compare**, **construct_version**, **create_branch**, **create_element**, **create_version**, **delete_branches_versions**, **merge**, **xcompare**, **xmerge**

DESCRIPTION

A *type manager* is a suite of programs that manipulates files with a particular data format; different type managers process files with different formats. A *directory type manager* provides programs that compare and/or merge versions of directory elements. ClearCase provides several type managers; users can create additional ones.

Several ClearCase version-control commands for file elements are implemented in two phases:

1. **Updating of the VOB database.** This phase is independent of the element's data format, and is handled directly by *cleartool*.
2. **Manipulation of the element's data.** In this phase, the data format is extremely significant, and so is handled by a particular type manager. *cleartool* invokes the type manager as a separate program, rather than as a subroutine. This provides flexibility and openness, allowing users to integrate their own data-manipulation routines with ClearCase.

For example, checkin of a *text_file* element involves (1) storing information in the VOB database about who created the new version, when it was created, and so on; (2) computing and storing the *delta* (incremental difference) between the new version and its predecessor. For a different type of element — for example, a bitmap file — the delta would be computed very differently, or not at all, and so would require a different type manager.

ELEMENT TYPES AND TYPE MANAGERS

Each file element type is associated with a type manager. For the predefined file element types, the associations are as follows:

Element Type	Type Manager	Purpose
file	whole_copy	store any data
compressed_file	z_whole_copy	store any data, using compress(1)
text_file	text_file_delta	store text, using incremental deltas
compressed_text_file	z_text_file_delta	store text, using both compress(1) and deltas
directory	directory	compare and merge directory versions

By default, an element type inherits the type manager of its supertype, but you can specify an alternative type manager when creating an element type. (See the `-supertype` and `-manager` options to *mkeltype*.)

USING A TYPE MANAGER

To have a particular file element use a particular type manager, you must establish two connections:

```
file element ----> element type ----> type manager
```

1. Make sure the VOB has an element type that is associated with the desired type manager. Use the `lstype –eltype –long` command to identify an existing element type. Alternatively, use the `mkeltype -manager` command to create a new element type that is associated with the desired type manager.
2. Create the file element, specifying the element type with the `–eltype` option. If the file element already exists, use the *chtype* command to change its element type.

You can automate the assignment of the new element type to newly-created elements using the ClearCase *file typing* facility, driven by *.magic* files. See the *cc.magic* manual page for details; see also the "Example" section below.

TYPE MANAGER STRUCTURE

A *type manager* is a collection of programs in a subdirectory of */usr/atria/lib/mgrs*; the subdirectory name is the name by which the type manager is specified with the `–manager` option in a *mkeltype* command.

Methods

Each program in a type manager subdirectory implements one *method* (data-manipulation operation). A method can be a compiled program, a shell script, or a link to an executable. It is invoked automatically at the appropriate time by a ClearCase version-control command.

A type manager can include these methods:

create_element	Invoked by *mkelem* to create an element's initial data container.
create_branch	Invoked by *mkbranch* to create a branch in an element's version tree.
create_version	Invoked by *checkin* to store a new version of an element.
annotate	Invoked by *annotate* to produce an annotated listing of a version's contents.
construct_version	Invoked by a view's *view_server* process when a file element is opened, from versions stored in delta or compressed format. This method constructs a readable, *cleartext* copy of a particular version. After the cleartext version is constructed, its line terminators may be adjusted by the *view_server*, according to the view's *text mode*. See the "Text Files, Cleartext, and a View's Text Mode" section in the *mkeltype* manual page, and the *mkview* manual page.
delete_branches_versions	Invoked by *rmver* and *rmbranch* to delete versions of an element.
delete_branches_versions	Invoked by *diff* and *xdiff* to run a file-comparison program that is specific to the element's data format.
delete_branches_versions	Invoked by *merge* and *xmerge* to run a file-merge program that is specific to the element's data format.

A type manager need not implement every method. For example, a type manager for bitmap graphics images may omit the *merge* method, because the operation doesn't make sense for that file format. In this case, the command `cleartool merge` will produce an error when invoked on an element that uses this type manager.

Method Inheritance and Links

A type manager can use symbolic links to *inherit* one or more of its methods from another type manager. For example, the type manager for *nroff*(1) source files described in the "Examples" section below is a variant of the standard *text_file_delta* type manager: it uses links to inherit all the *text_file_delta* methods, except for *compare*.

Another typical usage of symbolic links is to have individual methods be links to a "master" type manager program, which implements several (or all) of the methods. For an example, see directory */usr/atria/lib/mgrs/z_whole_copy*.

A link to the *cleardiff* program can implement the *compare* and/or *merge* method for text files. Similarly, a link to the *xcleardiff* program can implement the *xcompare* and/or *xmerge* method. Again, see directory */usr/atria/lib/mgrs/z_whole_copy* for an example.

Data Containers

Type managers process *data containers*, each of which stores the actual data for one or more versions of some element. All data containers are standard UNIX files, and are stored in the VOB's source pools, which are standard UNIX directories. Only type managers deal with data containers directly; users always manipulate data using the names of elements and links.

Performing the data manipulation for a version-control operation involves several programs. For example, to create a new version of an element:

1. *cleartool* generates the pathname (within a source pool) for a new data container.
2. On the VOB host (where the VOB storage area resides), a *vob_server* process creates an empty file at that pathname.
3. On the client host (where the user is working), the type manager fills the new data container with the data for the new version. (If the type manager implements deltas, it writes the data for one or more other versions to the new container, too.)
4. The *vob_server* changes the access mode of the new data container, making it unwritable.
5. Using the MGR_DELETE_KEEP_JUST_NEW exit status returned by the type manager, the *vob_server* deletes the old data container.

NOTE: Even with a type manager that implements deltas, a new data container is created each time a new version is created. In this case, the old container (which may have stored 27 versions) is replaced by the new container (which stores 28 versions). A type manager must *never* write to an old container or delete a old container (it usually won't have rights to do so).

CREATING A NEW TYPE MANAGER

You can create any number of new type managers for use throughout the local network. Use these guidelines:

1. Choose a name for the new type manager — ideally, one that shows its relationship to the data format (for example, *bitmap_mgr*). Create a subdirectory of */usr/atria/lib/mgrs* with this name.
2. Create symbolic links to make the new type manager inherit some of its *methods* (file-manipulation operations) from an existing type manager.
3. Create your own program for the method(s) that you wish to customize. See "Writing a Type Manager Program" below.
4. On each other ClearCase client host in the network, either make a copy of the new type manager directory, or create a symbolic link to it. The standard storage, performance, and reliability tradeoffs apply.

NOTE: An element type belongs to a VOB, and thus is available on every host that mounts its VOB. But a type manager is host-specific — it is some host's ***/usr/atria/lib/mgrs/****manager-name* directory.

Example

This section describes how the guidelines presented above can be used to create a type manager for elements that store a project's *nroff*(1) source files. We wish to implement the **compare** method with a program that compares the corresponding formatted files, instead of the source files. But we do not wish to change the **merge** method, since that is an operation to be performed on the source files themselves. Thus, the type manager:

- will be a refinement of the *text_file_delta* type manager
- will have the same functionality as *text_file_delta* for all methods except *compare*
- will compare two or more versions by first creating formatted pure-ASCII files with *nroff*(1), then using *cleardiff* to display the differences

The step numbers below correspond to those in the preceding section.

1. Create a directory for the type manager:

```
% su
<enter password>
# cd /usr/atria/lib/mgrs
# mkdir nroff_delta
```

2. For all the methods except *compare*, create symbolic links back to the *text_file_delta* directory:

```
# cd nroff_delta
# ln –s ../text_file_delta/construct_version construct_version
# ln –s ../text_file_delta/create branch create_branch
# ln –s ../text_file_delta/create_element create_element
# ln –s ../text_file_delta/create_version create_version
# ln –s ../text_file_delta/delete_branches_versions delete_branches_versions
# ln –s ../text_file_delta/merge merge
# ln –s ../text_file_delta/xcompare xcompare
# ln –s ../text_file_delta/xmerge xmerge
```

3. Create a *compare* program as an executable shell script. It might contain:

```
#!/bin/sh
#               read file that defines methods and exit statuses
. ${ATRIAHOME:-/usr/atria}/lib/mgrs/mgr_info.sh

OPTS=""
while (expr $1 : '\-' > /dev/null) ; do
    OPTS="$OPTS $1"
    if [ "$1" = "$MGR_FLAG_COLUMNS" ] ; then
        shift 1
        OPTS="$OPTS $1"
    fi
    shift 1
done

COUNT=1
for X in $* ; do
    nroff -man $X | col | ul -Tcrt > /usr/tmp/compare.$$.$COUNT
    COUNT=`expr $COUNT + 1`
done

echo Comparing files: $*
cleardiff -quiet $OPTS /usr/tmp/compare.$$.*
rm -f /usr/tmp/compare.$$.*
```

4. Create symbolic links to (or create copies of) the *nroff_delta* directory in other hosts' */usr/atria/lib/mgrs* directories. For example:

```
# rlogin saturn
# cd /usr/atria/lib/mgrs
# ln -s /net/neptune/usr/atria/lib/mgrs/nroff_delta nroff_delta
```

The *nroff_delta* type manager is now ready to be used.

1. Create the *manpage* element type, associating the new type manager with it:

```
# cleartool mkeltype –supertype text_file \
    –manager nroff_delta manpage
Comments for "manpage":
Variant of text_file, for nroff source files
.
Created element type "manpage".
```

2. Convert an existing element to type *manpage*:

```
% cleartool chtype –force manpage hello.1
Changed type of element "hello.1" to "manpage".
```

3. Compare two versions of the element, using the new type manager:

```
% diff hello.1@@/main/3 hello.1
7,8c7,8
< program outputs the message
< ''Hello, World''
---
> program sends the message
> ''Hello there, World''
```

```
% cleartool diff –serial hello.1@@/main/3 hello.1
Comparing files: hello.1@@/main/3 hello.1
```

```
-----[12 changed to 12]-----
<    The program outputs the message ‘‘Hello, World’’ to the
---
>    The program sends the message ‘‘Hello there, World’’ to the
```

4. Revise your personal *magic* file (for example, *$HOME/.magic/my.magic*) to have certain newly-created elements automatically get the *manpage* element type. For example:

   `manpage text_file : -name "*.[1-8]"`
   Classifies any file whose name ends with a digit suffix (*.1–.8*) as an *manpage* file.

   `manpage text_file : -name "manual_pages/*"`
   Classifies any file within the *manual_pages* directory as an *manpage* file.

WRITING A TYPE MANAGER PROGRAM

When invoking a type manager method, *cleartool* passes it all the arguments needed to perform the operation, in ASCII format. For example, many methods accept a *new_container_name* argument, specifying the pathname of a data container to which data is to be written.

In many cases, one or more of the parameters can be ignored. For example, the *create_version* method is passed *pred_container_name*, the pathname of the predecessor version's data container. If the type manager implements incremental differences, this is required information; otherwise, the predecessor's data container is of no interest.

Arguments are often ClearCase object identifiers (*OIDs*). You need not know anything about how ClearCase generates OIDs — just consider each OID to be a unique name for an element, branch, or version. In general, only type managers that store multiple versions is the same data container need be concerned with OIDs.

For more information on argument processing, see files */usr/atria/lib/mgrs/mgr_info.h* (for C language programs) and */usr/atria/lib/mgrs/mgr_info.sh* (for Bourne shell scripts), along with the chapter "Type Managers and Customized Processing of File Elements" in the *ClearCase User´s Manual*.

Exit Status of a Method

A user-defined type manager method must return an exit status to *cleartool*, indicating how the command is to be completed. The symbolic constants in */usr/atria/lib/mgrs/mgr_info.sh* specify all valid exit statuses. For example, an invocation of *create_version* might successfully create a new data container, then return exit status MGR_STORE_KEEP_JUST_NEW; if creation of the new data container fails, it would return exit status MGR_STORE_KEEP_JUST_OLD.

FILES

/usr/atria/lib/mgrs/*
/usr/atria/lib/mgrs/mgr_info.h
/usr/atria/lib/mgrs/mgr_info.sh

SEE ALSO

cleartool subcommands: mkelem, mkeltype
cc.magic, cc.icon, compress(1)

NAME version_selector – ClearCase version selector syntax

SYNOPSIS

branch-pathname/version-number

[*branch-pathname/*] *version-label*

[*branch-pathname/*] { *query* } (query must be enclosed in braces)

DESCRIPTION

A *version selector* identifies a version of an element in a version tree. You can use it with the `–version` command-line option in *cleartool*, as part of a rule in a *config spec*, and as part of a *version-extended pathname*. The version selector has three general forms. Each identifies a version in a different way:

- by *version-ID*
- by the version label attached to it
- by a query on the meta-data attached to it, or some other version characteristic

A version selector selects one version of an element, no version of an element, or generates an error, if ambiguous.

BRANCH PATHNAMES

All forms of the version selector may include a *branch pathname* to identify the branch on which a version resides. (Only one form *requires* a branch pathname — the *version-ID*; see next section). A branch pathname consists of a series of branch type names separated by slashes (`/`). A version tree has the same hierarchical structure as a directory tree; the root of this tree is the main branch (default name: */main*), which must always be the first entry in the branch pathname. The names of other branches follow the tree structure. Examples:

`/main`	*main branch*
`/main/bugfix`	*bugfix branch, off the main branch*
`/main/motif/bugfix`	*bugfix branch, off the /main/motif branch*
`/main/motif/bugfix/jpb`	*jpb branch, off the /main/motif/bugfix branch*

SELECTION BY VERSION-ID: branch-pathname/version-number

Selects the version with the specified version-ID. This is the only form that *requires* a branch pathname. Examples:

`/main/2`	*version 2 on main branch*
`/main/bugfix/5`	*version 5 on bugfix branch off main branch*
`/main/motif/bugfix/1`	*version 1 on subbranch of /main/motif branch*

In a version-extended pathname, the version-ID follows the element name and *extended naming symbol* (default: `@@`). For example:

`hello.c@@/main/4`	*version 4 on main branch of file 'hello.c'*
`util.c@@/main/motif/experiment/1`	*version 1 on the /motif/experiment branch, off the main branch*
`include@@/main/4/hello.h/main/3`	*version 3 on the main branch of file 'hello.h', in version 4 on the main branch of directory 'include'*

SELECTION BY VERSION LABEL: [branch-pathname/]label

Selects the version with the specified *version label*. The branch pathname is optional. Examples:

`/main/LATEST`	*most recent version on main branch*
`.../bugfix/REL2`	*version labeled REL2 on a branch named bugfix, at any branching level*
`/main/bugfix/REL2`	*version labeled REL2 on a bugfix branch that is a subbranch of main*
`/main/sunport/openlook/BUG3`	*version labeled BUG3 on a particular third–level branch*
`REL2`	*version labeled REL2 on any branch*

The label *LATEST* is predefined by ClearCase. It automatically evaluates to the most recent version on each branch of an element. If the most recent version on the main branch is version 4, then these two version selectors identify the same version:

```
/main/LATEST
/main/4
```

A version selector can consist of a *standalone* label, such as *REL2*. Standalone labels can be ambiguous, however. For example, */main/bugfix/REL2* and *REL2* may or may not be equivalent for a given element:

- If the *REL2* label type was created as one-per-element (default), the two version selectors must be equivalent.
- If *REL2* was created with `mklabel -pbranch`, however, the label can be used once per branch. If the label is actually attached to two or more versions of an element, an `ambiguous` error occurs. No error occurs for elements that happen to have only one instance of a one-per-branch label type.

Version Labels As Hard Links

Version labels appear as UNIX hard links in an element's directory tree in *version-extended namespace*. (See the *pathnames_ccase* manual page.) If a version label was defined to be one-per-element, then an additional hard link appears at the top level of an element's directory tree. For example, if *BL3* is a one-per-element label, then these version-extended pathnames are both unambiguous references to the same version:

```
hello.c@@/BL3
hello.c@@/main/bugfix/patch2/BL3
```

In effect, this feature allows you to reference a version without knowing its exact location in the version tree.

If a label was defined with the `–pbranch` option, it does not appear in the element's top-level extended namespace directory (as implied above). Thus, if the one-per-element label, *BL3*, and the one-per-branch label, *TEST_LBT*, was attached to version */main/1* of file *hello.c*, its top-level extended namespace directory would look like this:

```
% cd hello.c@@
% ls
BL3  main
```

SELECTION BY QUERY: [branch-pathname/]{query}

Selects the version that satisfies the specified query. A branch pathname is optional.

The query expression consists of one or more query primitives and operators, organized according to the syntax rules listed in the *query_language* manual page. The query expression must be enclosed in braces ({ ... }), and the entire version selector in single quotes (' ... ') or double quotes (" ... ") if it includes spaces, or characters that have special meaning to the shell. String literals within the query expression must be double-quoted.

Examples:

`/main/{TESTED=="yes"}`	*the latest version on the main branch for which the 'TESTED' attribute has the value 'yes'*
`{hltype(design_spec,<-)}`	*the version on any branch that is the 'to' end of a hyperlink of type 'design_spec'*
`/main/bugfix/{!lbtype(REL2)}`	*the latest version on the bugfix branch which is not labeled 'REL2'*
`{created_by(jpb)&&pool(sr1)}`	*the version on any branch created by user 'jpb' which is stored in the 'sr1' storage pool*

If the version selector includes a branch pathname, the *view_server* selects the *latest* version on the branch that satisfies the query. If the version selector does not include a branch pathname, the *view_server* selects the version on *any* branch that satisfies the query. However, without a branch pathname, a query is ambiguous when more than one version of the element satisfies the query; versions on different branches, or two versions on the same branch, for example.

An ambiguous query fails to select any version. A query will also fail if no version satisfies the query.

A version-extended pathname can include a query, but is subject to the same restrictions as other version selectors of this form. That is, the query must select exactly one version to succeed. For example, this command displays the most recent version that has an attribute of type *TESTED*:

```
% cat include.h@@/"{attype(TESTED)}"
```

Note the use of quotes to prevent the shell from interpreting the curly brace and parenthesis characters. As an alternative, you can quote the entire pathname:

```
% cat "include.h@@/{attype(TESTED)}"
```

If multiple branches have versions with a *TESTED* attribute, the version-selector used in the examples above is ambiguous, and an error occurs.

RESTRICTION: In a version-extended pathname, you cannot use both a branch pathname and a query:

```
% cat "include.h@@/main/{attype(TESTED)}"                (does not work)
% cat "include.h@@/main/rel2_bugfix/{attype(TESTED)}"    (does not work)
```

You can use the *describe* command to work around this restriction:

```
% cat `cleartool describe –s –ver /main/rel2_bugfix/"{attype(TESTED)}" include.h`
```

SEE ALSO

cleartool, config_spec, pathnames_ccase, query_language

NAME view – ClearCase view data structures

SYNOPSIS

UNIX directory tree created by 'mkview' command

DESCRIPTION

A *view* provides a virtual workspace, in which users can access shared data, stored in one or more VOBs, and private data, which is stored in the view itself. For users to access a view on a ClearCase client host, the view must be activated on that host (for example, with *setview*).

This manual page discusses both a view's physical data structures and the way file system data appears to a user process through a view.

VIEW STORAGE DIRECTORY

A view is implemented as a standard directory tree, whose top-level directory is termed the *view storage directory*. The directory contains files and subdirectories:

.pid A one-line text file that lists the process-ID of the associated *view_server* process, currently running on the host where the view storage directory resides.

.view A file that lists the view's "universal unique identifier" (*UUID*):

```
neptune:/home/akp/tut/old.vws
26a7d404.428211cd.b405.08:00:69:06:af:65      (view's UUID)
neptune
```

The other lines are historical artifacts; they indicate where the view was originally created, but are not used in the current ClearCase release to determine a view's current location. (This is information is maintained in the network's *storage registry*.)

config_spec A file that stores the view's current config spec, in the form displayed by *catcs*.

.compiled_spec
A modified version of *config_spec*, which includes ClearCase-internal accounting information.

.identity A subdirectory whose files establish the view's owner and group memberships. Only the view's creator has any access rights to this subdirectory. It has the same structure as the like-named subdirectory within a VOB storage directory. See the *vob* manual page for more information.

.s A subdirectory that implements the view's *private storage area*. See "Private Storage Area" below.

db A subdirectory containing the files that implement the view's embedded database. See "View Database" below.

Private Storage Area

Subdirectory *.s* of the view storage directory is the root of a subtree that implements the view's *private storage area*. If the view was created with `mkview -ln`, then *.s* is actually a (UNIX-level) symbolic link, pointing to a remote storage area.

The private storage area holds several kinds of objects:

View-Private Objects. A *view-private object* is a file system object — file, directory, or link — created by a standard program within a VOB directory. Such objects are stored only within the view's private storage area; no VOB maintains any record of such objects.

Checked-Out Files. A *checked-out version* is a file created by the *checkout* command. This file is an editable copy of the version being checked out.

A checked-out version is very much like a view-private file, except that there is a corresponding object in the VOB database: the special "placeholder" version with the *CHECKEDOUT* version label.

Unshared Derived Objects. An *unshared derived object* is a data container created by execution of a *makefile* build script by *clearmake*, or by any program invoked by *clearaudit*. A corresponding *derived object* is created in the VOB database.

NOTE: An unshared derived object remains in the view's private storage area even after the DO becomes shared; *promotion* of the DO involves a *copy* — not a move — of the data container. The *winkin* command and *view_scrubber* utility remove unshared derived objects from a view's private storage area.

Configuration Records. File *view_db.crs_file* in the *.s* subdirectory is actually part of the view's database (see below): the part that stores the configuration records of derived objects built in the view.

View Database

The view database subdirectory, *db*, contains these files:

view_db.dbd A compiled database schema, used by ClearCase's embedded DBMS routines for database access. The *schema* describes the structure of the view database. The *mkview* command creates this file by copying */usr/atria/etc/view_db.dbd*.

view_db_schema_version
A schema version file, used by ClearCase's embedded DBMS routines to verify that the compiled schema file is at the expected revision level. The *mkview* command creates this file by copying */usr/atria/etc/view_db_schema_version*.

*view_db.d0**n***
view_db.k01 Files in which the database's contents are stored.

*vista.** Database control files and transaction logs.

view_db.crs_file Stores the configuration records of unshared derived objects. As described above, this file resides in subdirectory *.s* of the view storage directory, allowing it to be remote.

The view database keeps track of the objects in its private storage area: view-private objects (files, directories, and links), checked-out versions, and unshared derived objects.

SEE ALSO

cleartool subcommands: lsview, mkview, setview, startview, lsview
mvfsstorage, registry_ccase, scrubber, view_scrubber, vob
ClearCase Administrator´s Manual

NAME view_scrubber – remove derived object data containers from view storage

SYNOPSIS

/usr/atria/etc/view_scrubber [**–p**] [**–k**] [**–n**] [*DO-pname* …]

DESCRIPTION

WARNING: This command modifies the way in which view-resident objects are combined with VOB-resident objects to produce a 'virtual workspace'. To avoid errors, make sure that no application or development tool is using the view's files when this command is executed.

The most common way to run the *view_scrubber* is indirectly, by running the */usr/atria/etc/view_scrubber.sh* script supplied with ClearCase.

The *view_scrubber* program "cleans" a view's private storage area by removing derived object data containers. This is useful in these situations:

Scenario 1: Cleaning Up after a 'clearmake' Wink-In. The first time it *winks-in* a derived object, *clearmake* copies the data container from private storage (of the view where the object was originally built) to shared storage (a VOB storage pool). This procedure, termed *promotion*, involves a *copy*, not a *move*. (See the *promote_server* manual page for details.) At this point:

- The view where the derived object was originally built continues to use the data container in view storage.
- Any other view into which the derived object is subsequently winked-in uses the promoted data container in VOB storage.

Running *view_scrubber* in the view where the derived object was built simplifies the situation. First, it uses *rm*(1) to remove the derived object; this deletes the data container from view storage. Then, it restores the derived object through a wink-in; the establishes a link to the data container in VOB storage.

Now, all views that share the derived object access the data container in VOB storage; and the redundant, space-consuming data container in view storage has been eliminated.

Scenario 2: 'Self-Wink-In'. By default, the data container for an unshared derived object always remains in view storage (until the DO is deleted or overwritten). `view_scrubber -p` transfers the data container to VOB storage, thus freeing space in the view storage area. In essence, this involves winking-in the derived object to the same view. First, the data container is promoted from view storage to VOB storage; then, the remove-then-restore procedure described in Scenario 1 is invoked.

NOTE: Scenario 2 can also be accomplished with the *winkin* command.

OPTIONS AND ARGUMENTS

Preprocessing with a 'Promotion'. *Default: view_scrubber* removes view-resident data containers, then restores the derived objects to the view through wink-in. Requirement: the derived objects' data containers must already be in VOB storage.

–p Before performing the default processing described above, *promotes* (copies) the derived objects' data containers from view storage to VOB storage. This removes the requirement noted above.

Error Recovery. *Default: view_scrubber* aborts if it is unable to complete its work on any derived object.

–k Keeps going, even if one or more derived objects cannot be successfully processed.

No-Execute Option. *Default: view_scrubber* performs its work and displays appropriate messages.

–n Suppresses the actual processing of data containers; messages are displayed to indicate what work *view_scrubber* would have performed.

Derived Objects to Process. *Default:* If you don't specify any derived objects as command arguments, *view_scrubber* reads a one-per-line list of pathnames from *stdin*, which must be a pipe.

DO-pname …
One or more standard pathnames of derived objects.

EXAMPLES

- Make the view you wish to scrub the current working view, and move to the directory of interest. Then scrub DO containers for the entire directory tree, using the script */usr/atria/etc/view_scrubber.sh* (which invokes the *view_scrubber* program).

```
% cleartool setview big_view
% cd /vobs/src
% /usr/atria/etc/view_scrubber.sh
```

SEE ALSO

cleartool subcommands: winkin
clearmake, promote_server, scrubber

NAME view_server – server process that performs version selection for a view

DESCRIPTION

A *view_server* is a long-lived process that manages activity in a particular ClearCase *view*. It interprets the rules in the view's *config spec*, and tracks modifications to view-private files for other ClearCase software.

Each view requires a dedicated *view_server* on the host where the view storage area resides. The *view_server* is started by its host's *albd_server* process when necessary. It runs with the user-ID of the owner of the view storage directory (usually, the user who created the view). A *view_server* remains active until it is terminated by a system shutdown or a *kill*(1) command.

A *view_server* handles MVFS file system requests (such as create, delete, and rename) by querying one or more *VOB databases* and comparing them against the view's own database. Using the view's config spec, it selects versions of file elements and directory elements to appear in the view. It also handles requests from *cleartool*, *clearmake*, and *clearaudit* to look up VOB-database objects and/or names.

A *view_server* manages its view's database by tracking changes to view-private objects against the related objects in VOB databases (for example, the view-private file that corresponds to the checked-out version of a file element).

VIEW CONFIGURATION

When it begins execution, a *view_server* reads configuration information from file *.view* in the view storage directory. This is an ASCII file, which contains:

- *line 1:* the location of the view storage directory, in *hostname*:*pathname* format
- *line 2:* the view's UUID (unique identifier), which must not be changed
- *line 3:* the *hostname* specified in line 1

NOTE: Lines 1 and 3 are placed in the *.view* file when the view is created, but the *view_server* ignores these lines thereafter.

The configuration file can include additional entries, each on a separate line:

–cache *size-in-bytes*
Sets the total size of the *view_server*'s caches to be *size-in-bytes*. The default is 204800 (200Kb). This total is allocated among the several caches automatically.

–readonly Prevents modification of the view's private data-storage area. A read-only view cannot be used for checkouts or for builds, since these operations create new files in view-private storage. (A *checkout* command will succeed in creating a checked-out version in the VOB database, but will not be able to create the corresponding view-private file.)

NOTE: `–readonly` does not prevent users from changing a view's *config spec*. Use view access permissions for to implement this kind of restriction. (See the *mkview* manual page for details.)

The following example shows how a view's owner can reconfigure a view to double its cache allocation. Suppose the view storage directory is located at *$HOME/jones_3.vws*, and is registered with view-tag *jones_3*.

```
% cd $HOME/jones_3.vws                    (go to view storage directory)
% chmod +w .view                          (make configuration file writable)
% echo "-cache 409600" >> .view           (add option line to configuration file)
% chmod 444 .view                         (restore permissions of configuration file)
% kill 'cat .pid'                         (kill view_server, using stored PID)
% cleartool startview jones_3              (restart view_server)
```

VIEW PERFORMANCE STATISTICS

This command causes a *view_server* to write cumulative cache-performance (and other) statistics to its log file, */usr/adm/atria/log/view_log*:

kill -HUP *view_server-process-ID*

It then resets all the statistics accumulators to zero. You must enter this command on the host where the *view_server* executes.

SEE ALSO

cleartool subcommands: mkview, mktag, recoverview, reformatview, rmtag, rmview, setview, startview, albd_server, clearmake, kill(1)

NAME VOB – ClearCase VOB data structures

SYNOPSIS

UNIX directory tree created by 'mkvob' command

DESCRIPTION

A *VOB* (*versioned object base*) is a secure data repository for a directory tree. For users to access a VOB on a ClearCase client host:

- The VOB must be *activated* on the host by mounting it as a file system of type MVFS (ClearCase's *multiversion file system* type).
- The VOB must be accessed through a *view*.

This manual page discusses both a VOB's physical data structures and its logical structures, as seen by a user process through a *view*.

PHYSICAL DATA STRUCTURES / VOB STORAGE

A VOB is implemented as a standard directory tree, whose top-level directory is termed the *VOB storage directory*. The directory contains files and subdirectories:

.pid A one-line text file that lists the process-ID of the associated *vob_server* process, currently running on the host where the VOB storage directory resides.

vob_oid A one-line text file that lists the VOB's "universal unique identifier"(*UUID*). ("OID" means "object identifier".) or OID. This UUID is the same for all the *replicas* in a *VOB family* (Atria MultiSite product).

replica_uuid A one-line text file that lists the ClearCase-internal *replica UUID* of this particular replica of the VOB. Different replicas created with the Atria MultiSite product have different identifiers.

.identity A subdirectory whose files establish the VOB's owner and group memberships. See "The .identity Directory" below.

s A subdirectory in which all of the VOB's local *source storage pools* reside.
d A subdirectory in which all of the VOB's local *derived object storage pools* reside.
c A subdirectory in which all of the VOB's local *cleartext storage pools* reside.

See "VOB Storage Pools" below.

db A subdirectory containing the files that implement the VOB's embedded database. See "VOB Database" below.

The .identity Directory

Subdirectory *.identity* records the VOB's ownership and group membership information. This directory has a very restricted access mode: only the user who is the *VOB owner* has any access rights. (As always, the host's *root* user can also access this directory.)

The *.identity* directory contains these files:

uid The owner of this file is the *VOB owner*.

gid This group to which this file belongs is the VOB's *principal group*.

*group.**nn*** Each additional file (if any) indicates by its group membership an additional group on the VOB's group list. In addition, the file's name identifies the group by numeric ID (*group.30*, *group.2*, and so on).

You can use a `describe -vob` command to display this information.

Example: A VOB is created by user *drp*, whose password entry places him in the *vga* group, and who also belongs to groups 2 (*bin*) and 30 (*dvt*). The VOB storage directory's *.identity* subdirectory contains these files to record this information:

```
-r----S---   1 drp      vga     0 Oct  9 09:34 gid
-r----S---   1 drp      bin     0 Oct  9 09:34 group.2
-r----S---   1 drp      dvt     0 Oct  9 09:34 group.30
-r-S------   1 drp      vga     0 Oct  9 09:34 uid
```

User *drp* or *root* subsequently adds group 50 (*rlsgrp*) to the VOB's group list:

```
% cleartool protectvob –add_group rlsgrp /vobstore/...
```

This change is recorded by an additional file in *.identity:*

```
-r----S---   1 drp      rlsgrp  0 Oct  9 09:34 group.50
```

VOB Storage Pools

Each VOB storage directory is created with three default *storage pools*, located within the directories listed above.

s/sdft Default source storage pool, for permanent storage of versions' file system data.

c/cdft Default cleartext storage pool, for temporary storage of the *cleartext* versions currently in use (for example, reconstructed versions of *text_file* elements).

d/ddft Default derived object storage pool, for storage of shared derived objects.

For more information on storage pools, see the *mkvob*, *mkpool*, and *chpool* manual pages.

VOB Database

The VOB database subdirectory, *db*, contains these files:

vob_db.dbd A compiled database schema, used by ClearCase's embedded DBMS routines for database access. The *schema* describes the structure of the VOB database. The *mkvob* command creates this file by copying */usr/atria/etc/vob_db.dbd*.

vob_db_schema_version
A schema version file, used by ClearCase's embedded DBMS routines to verify that the compiled schema file is at the expected revision level. The *mkvob* command creates this file by copying */usr/atria/etc/vob_db_schema_version*.

*vob_db.d0**n***
*vob_db.k0**n*** Files in which the database's contents are stored.

*vista.** Database control files and transaction logs.

db_dumper A copy of */usr/atria/etc/db_dumper*. This is an executable program, invoked during the *reformatvob* command's "dump" phase. Each VOB gets its own copy of *db_dumper* so that it will always be able to "dump itself" to ASCII files. (Typically, it will need to be dumped after a newer release of ClearCase has already been installed on the host; with this strategy, the */usr/atria/etc/db_dumper* program in the newer release need not know about the "old" VOB database format.)

Backup Database Subdirectories. *reformatvob* does its work by creating a new VOB database. By default, it preserves the old database by moving it aside to a date-stamped name. Thus, a VOB storage directory may contain old, (usually) unneeded VOB database subdirectories, with names like *db.0318*. If *reformatvob* is interrupted, it may leave a partially-reformatted database with the name *db.reformat*.

LOGICAL DATA STRUCTURES

From the user's standpoint, a VOB contains *file system objects* and *meta-data*. Some meta-data is stored in the form of objects; other meta-data is stored as records or annotations attached to objects.

'VOB-Itself' Object / Replica Objects

Each VOB database contains one object that represents the VOB itself. Termed the *VOB object*, it provides a "handle" for certain operations — for example:

- listing event records of operations that affect the entire VOB (`lshistory -vob`). This includes creation and deletion of type objects, removal of elements, and so on.
- Placing a lock on the entire VOB (`lock -vob`).

Using the Atria MultiSite product, you can create any number of *replicas* of a VOB at different sites. Each VOB replica is represented in the VOB database by a *replica object*.

File System Objects

A VOB database keeps track of users' file system objects using the following database objects:

file element An object with a *version tree*, consisting of *branches* and *versions*. Each version of a file element has *file system data*: a sequence of bytes. Certain element types constrain the nature of the versions' file system data; for example, versions of *text_file* elements must contain text lines, not binary data.

directory element
An object with a *version tree*, consisting of *branches* and *versions*. Each version of a directory element catalogs a set of file elements, directory elements (subdirectories), and *VOB symbolic links*. An "extra" name for an element this is already entered in some other directory version is termed a *VOB hard link*.

VOB symbolic link
An object whose contents is a text string. This string is interpreted by standard commands in the same way as an operating system symbolic link.

Link Counts for File System Objects. Link counts for file system objects are stored in the VOB database, and reported by the standard *ls*(1) command, as follows:

symbolic link	1
file element	1
file version	1
directory element	2
directory version	2 plus number of directory elements cataloged in that version
branch	2 plus number of subbranches (each branch appears as a subdirectory in the extended-namespace representation of an element's version tree)

This scheme satisfies two UNIX rules:

- The link count is at least one for an object that has a name.
- The link count of a directory is 2 + number-of-subdirectories.

The scheme does not satisfy the rule that the link count should be the number of names the object has in the current namespace.

Type Objects

A VOB can store several kinds of *type objects*:

element type	Defines a class of elements within the VOB.
branch type	Defines a set of like-named branches in some or all of the VOB's elements.
label type	Defines a mnemonic name that can be attached to a set of versions, thus defining a configuration of the VOB's elements.
attribute type	Defines a name to be used in attaching *name/value* pairs to VOB-database objects.
hyperlink type	Defines a class of logical arrow that can be used to connect pairs of objects.
trigger type	Defines a monitor on operations that modify the VOB's objects.
replica type	Defines a class of VOB replicas (Atria MultiSite product).

Instances of Type Objects

After a type object is created, users can create any number of *instances* of the type.

element	Each file or directory element in a VOB is created by *mkelem* or *mkdir* as an instance of an existing element type in that VOB.
branch	Each branch in an element is created by *mkbranch* as an instance of an existing branch type in that element's VOB.
version label	The *mklabel* command annotates a version with a version label, by creating an instance of an existing label type.
attribute	The *mkattr* command annotates a version, branch, element, VOB symbolic link, or hyperlink with an attribute, by creating an instance of an existing attribute type. Each instance of an attribute has a particular value — a string, an integer, and so on.

hyperlink The *mkhlink* command creates a hyperlink object, which is an instance of an existing hyperlink type. A typical hyperlink connects two objects, in the same VOB or in different VOBs.

trigger The *mktrigger* command creates a trigger object, which is an instance of an existing trigger type. The trigger becomes attached to one or more elements.

replica The Atria MultiSite product's *mkreplica* command creates a *VOB replica* object, which is an instance of an existing replica type.

Derived Objects

A VOB's database stores information on all the derived objects (DOs) created at pathnames within the VOB. For each DO, the database catalogs:

- the directory element, along with the location of the DO within the directory (for example, *util.o* or *sun5/util.o*)
- the DO's unique identifier, its *DO-ID*

See the *derived_object* manual page for more information.

Configuration Records

A VOB's database stores the *configuration records* (CRs) associated with derived objects and *DO versions* (derived objects that have been checked in as versions of elements). Each CR documents a single *target rebuild*, which typically involves execution of one build script. See the *config_record* manual page for more information.

Event Records

ClearCase creates an *event record* in the VOB database for nearly every operation that modifies the VOB. See the *events_ccase* manual page for more information.

Unneeded event records are periodically deleted from a VOB's database by the *vob_scrubber* utility.

Initial VOB Contents

When first created by the *mkvob* command, a VOB appears to users (through its type-MVFS file system mount point) as an almost-empty directory tree. It contains no files, and just two directories:

VOB Root Directory. *mkvob* automatically performs a *mkdir* command to create a directory element, the *VOB root directory*, in the new VOB. Mounting a VOB makes its root directory accessible at the *VOB-tag* (VOB mount point) pathname.

For most purposes, the VOB root directory is like any other ClearCase directory element you subsequently create within the VOB. But there are differences in certain contexts:

- The filename pattern in a config spec rule cannot be a relative pathname that begins at a VOB root directory. A relative pathname must start below a VOB root directory. See the *config_spec* manual page for details.

- You must use a special syntax for a version-extended name that specifies a location in the version tree of a VOB's root directory:

```
ls /usr/src/proj@@/main/3          (invalid if directory 'proj' is VOB root)
ls /usr/src/proj/.@@/main/3        (valid)
```

The VOB root directory is assigned to the three default storage pools. All newly-created file and directory elements will be assigned to the default storage pools until new pools are created and assigned.

The lost+found Directory. *mkvob* also creates a special directory element, *lost+found*, as a subdirectory of the VOB root directory. ClearCase places elements that are no longer cataloged in any directory version in this special directory. This occurs when you:

- create new elements, and then *uncheckout* the directory in which they were created
- delete the last reference to an element with the *rmname* command
- delete the last reference to an element by deleting a directory version with the *rmver*, *rmbranch*, or *rmelem* command

When an element is moved to *lost+found*, it gets a name of the form:

element_leaf_name.id-number.

The id-number is a unique (and rather lengthy) hexadecimal number, such as
41a00000bcaa11caacd0080069021c7.

The *lost+found* directory has several unique properties:

- It cannot be checked out.
- It *can* be modified without being checked out.
- No branches can be created within it.

To conserve disk space, purge the *lost+found* directory of unneeded elements on a periodic basis, using the *rmelem* command.

VOB REGISTRY AND VOB ACTIVATION

Each VOB is registered in the network-wide *storage registry*, as described in the *registry_ccase* manual page. The `cleartool mount` command activates a registered VOB by mounting it as a type-MVFS file system. See the ClearCase *mount* manual page for details.

SEE ALSO

cleartool subcommands: chpool, lsvob, mkvob, mkdir, mkelem, mkpool
config_record, config_spec, crontab_ccase, derived_object, events_ccase, mvfsstorage, registry_ccase, scrubber, view, vob_scrubber
ClearCase Administrator´s Manual

NAME vob_scrubber – remove event records from VOB database

SYNOPSIS

/usr/atria/etc/vob_scrubber [**–stats_only**] [**–long**] [**–nlog**]
{ **–lvobs** | *vob-storage-dir-pname* … }

DESCRIPTION

The *vob_scrubber* utility program deletes old *event records* (abbreviated to "events" in this manual page) from a VOB database. This retards VOB growth by logically deleting the events, freeing space in the VOB database for storage of new events. (Physical deletion requires processing with the *reformatvob* command.)

You can explicitly run *vob_scrubber* as needed; by default, it is automatically run periodically by *cron*(1M) — see "Crontab Processing" below. A configuration file, *vob_scrubber_params*, provides control over which events are deleted.

vob_scrubber does not need to run in a view and does not require the VOB(s) it processes to be mounted.

ClearCase Events

ClearCase creates a meta-data item called an *event* in a VOB database (almost) every time it modifies the database — for example, to record the checkin of a new version, the attaching of an attribute to an element, or the creation of a new branch type. Each event consumes 300–400 bytes. Some events, like those for element and version creation, are valuable indefinitely; however, many *minor events* are not. For example, the removal of a version label from a collection of versions creates a minor event for each affected object. Over time, such minor events occupy more and more space, while becoming less and less useful. (After a month or a year, no one is likely to care who removed the version labels, especially if the label type itself has also been deleted.)

Event Scrubbing

vob_scrubber marks certain events as logically deleted. As with any meta-data removal, the deletion does not physically reduce the amount of disk space used by the VOB database; it merely frees up space in the database, making it available for future use. To actually reduce the size of the database, you must run *reformatvob*, which discards the logically deleted data as it reconstructs the VOB database. Thus, regular use of *vob_scrubber* minimizes VOB database growth, but does not recover disk space.

What Events Are Deleted

These events are never deleted:

- The most recent 1000 events physically added to the VOB (regardless of logical event time). These are needed by views for cache invalidation.
- If an object is locked, the object's most recent lock event
- Events for operations not listed in section "VOB-Specific Event Scrubbing Parameters" below. (See the *events_ccase* manual page for a complete list of the operations that cause event records to be stored in the VOB database.)

These obsolete events are always deleted, regardless of event scrubbing parameters:

- Creation events for derived objects
- Events whose operation is mklabel, mkattr, mkhlink, mktrigger, rmlabel, rmattr, rmhlink, or rmtrigger (if the type object associated with the event has been deleted with *rmtype*)
- Events with event kind `destroy`, `modify`, or `modify meta-data` that have no generated comment. (Only VOBs created with very early releases of ClearCase have such events.)

All other events are preserved or deleted according to the configuration file specifications described in section "VOB-Specific Event Scrubbing Parameters" below.

Crontab Processing

ClearCase installation adds an entry to the *root* user's *crontab*(1), enabling once-a-week execution of the script */usr/atria/config/cron/ccase_cron.wk*. This script, in turn, invokes */usr/atria/config/cron/vob_scrubber.sh* to run *vob_scrubber* on each VOB that resides on the local host.

OPTIONS AND ARGUMENTS

Report Format and Destination. *Default:* Event statistics are listed briefly, with events categorized by kind of object (for example, all events for branch objects are grouped); report is sent to the standard log file, */usr/adm/atria/log/vob_scrubber_log*.

–long Produces a detailed report of the event statistics, with events categorized by kind of object, kind of event, and kind of operation.

–nlog Sends the report to *stdout* instead of the log file.

Deletion Control. *Default:* Delete events and report statistics on the number of objects, the number of events before deletion, the number of events deleted, and the number of events after deletion.

–stats_only
Suppresses event deletion; the report includes statistics on the number of objects and events in the VOB.

VOBs to be Processed. *Default:* None.

–lvobs Event-scrubs all mounted VOBs that reside on the local host.

vob-storage-dir-pname
Scrubs the VOB whose storage directory is at the specified pathname.

VOB-SPECIFIC EVENT SCRUBBING PARAMETERS

A system-wide event-scrubbing configuration file controls the operation of *vob_scrubber*; each VOB can have its own configuration file, which overrides the system-wide settings:

system-wide config file	**/usr/atria/config/vob/vob_scrubber_params**
per-VOB config file	*vob-storage-dir-pname***/vob_scrubber_params**

The event-scrubbing configuration file is a text file. A line that begins with a pound-sign character (#) is a comment. All other lines control how one kind of event is to be scrubbed — how long to keep the most recent one, and how long to keep other events of that kind:

operation **–keep_all** { *n* | **forever** } **[–keep_last** { *n* | **forever** } **]**

The components of an event-scrubbing control line are:

operation Kind of event, specified by the operation that creates the event. (See the table in the **–keep_last** description below.)

–keep_all { *n* | **forever** }
For each object: keep events created by the specified operation for at least *n* days, or forever. If `–keep_last` is also specified, this period applies to all but the most recent such event; otherwise, the period applies to all such events, including the most recent one.

–keep_last { *n* | **forever** }
(optional) For each object: keep the most recent event created by the operation for at least *n* days, or forever. The "keep_last" period must be at least as long as the "keep_all" period. The meaning of "most recent event" depends on the operation:

mklabel, **mkattr**, **mkhlink**, **mktrigger**
rmlabel, **rmattr**, **rmhlink**, **rmtrigger**
Most recent "modify meta-data" event for each type object on each element. (Example: for each element, the most recent event that records the attaching of version label *REL3* to some version)

lnname, **rmname**
Most recent "modify directory version" event for each directory element.

rmelem, **rmpool**, **rmtype**
Most recent event in VOB.

rmbranch, **rmver**, **chpool**, **chtype**
Most recent event on each element.

protect Most recent event on each element or derived object.

modpool Most recent event on each pool.

modtype Most recent event on each type.

lock, **unlock**
Most recent event on each object. Exception: if an object is locked, the most recent lock event on that object is never deleted.

Operation Log Scrubbing

A slightly different syntax controls scrubbing of *operation log* entries, produced by the Atria MultiSite product to synchronize VOB replicas:

oplog –keep { *n* | **forever** }

This control line specifies how long (in days) an operation log entry will be retained in the VOB database. Be sure to preserve operation log entries long enough to guarantee delivery of synchronization updates based on them.

Event-Scrubbing Defaults

If the configuration file includes no entry for an operation, all events created by the operation are kept forever. Hence, an empty configuration file preserves all events (except obsolete events, which are always discarded; see "What Events Are Deleted" above). The calculated times are always compared against the logical event creation time (as shown by *lshistory*), rather than the physical event creation time. These can differ if the events were created by a converter.

EXAMPLES

- For *unlock* events in all VOBs on the local host: keep the event if it occurred within the last 7 days (but 30 days for the most recent such event on a particular object); otherwise, delete it.

  In '/usr/atria/config/vob/vob_scrubber_params':

  ```
  unlock -keep_all 7 -keep_last 30
  ```

FILES

/usr/atria/config/vob/vob_scrubber_params
/usr/adm/atria/log/vob_scrubber_log

SEE ALSO

cleartool subcommands: reformatvob, lshistory
events_ccase, scrubber

NAME vob_server – ClearCase server program for VOB storage pool access

SYNOPSIS

invoked as needed by the 'albd_server' program

DESCRIPTION

For each VOB, a long-lived *vob_server* process runs on the VOB host, with the user-ID of the *VOB owner* (see *protectvob*) This process maintains the VOB's *storage pools* in response to requests from ClearCase client processes. This includes creating, deleting, and controlling the UNIX-level permissions of the pools' data containers.

The *vob_server* is the only process that ever creates or deletes data containers; the VOB owner is the only user who can modify data containers and storage pools. These severe restrictions protect VOB data against careless or malicious users.

A *vob_server* process is started automatically, as needed, by *albd_server*. It remains active until the operating system is restarted or the VOB is deleted with the *rmvob* command.

ERROR LOG

The *db_server* sends warning and error messages to */usr/adm/atria/log/vob_log*.

SEE ALSO

abe, albd_server, init_ccase, scrubber, view_server, clearmake, cleartool, nfsd(1M)

NAME vobrpc_server – ClearCase database server program

SYNOPSIS

invoked as needed by the 'albd_server' program

DESCRIPTION

Each VOB host runs up to five *vobrpc_server* processes for each of its VOBs. Each such process handles requests from *view_server* processes throughout the network. The request can generate both meta-data (VOB database) and file system data (storage pool) activity: the *vobrpc_server* accesses the VOB database in exactly the same way as a *db_server*; it forwards storage pool access requests to the *vob_server*.

vobrpc_server processes are managed similarly to NFS daemons (see *nfsd*(1M)). Multiple server processes are started automatically by *albd_server*, which also routes new requests to the least-busy servers, and terminates unneeded *vobrpc_server* processes when the system is lightly loaded.

ERROR LOG

The *db_server* sends warning and error messages to */usr/adm/atria/log/vobrpc_server_log*.

SEE ALSO

abe, albd_server, init_ccase, scrubber, view_server, clearmake, cleartool, nfsd(1M)

NAME wildcards_ccase – pattern-matching characters for ClearCase pathnames

SYNOPSIS

? * ~ ~*username* […] . . .

DESCRIPTION

The *wildcard* (or *pattern-matching*) characters listed below are recognized in these ClearCase contexts:

- **'cleartool' commands** — If you use *cleartool* in *single-command* mode, pathnames you specify on the command line are interpreted — and wildcards expanded — by the operating system shell, not by *cleartool*. In *interactive* mode, *cleartool* itself interprets the pathnames and expands wildcards.

  With some commands, you can specify a pathname or pathname pattern as a quoted argument:

  ```
  cleartool catcr -select 'bug?.o' bgrs@@04-Mar.22:54.426
  ```

  Such quoted pathname patterns are always interpreted by *cleartool*.

- **Config spec rules** — The pathname pattern in a config spec rule is interpreted by a view's associated *view_server* process.

Wildcard Characters

ClearCase software recognizes these wildcard characters:

?	Matches any single character.
*	Matches zero or more characters.
~	Indicates your home directory (even if you are not a C shell user).
~*username*	Indicates *username*'s home directory (even if you are not a C shell user).
[*xyz*]	Matches any of the listed characters.
[*x*–*y*]	Matches any character whose ASCII code falls between that of *x* and that of *y*, inclusive.

...	(ellipsis, a ClearCase extension) matches zero or more directory levels.
	Example 1: `foo/.../bar` matches any of the following pathnames:
	`/vobs/foo/bar`
	`/vobs/foo/usr/src/bar`
	`/vobs/foo/rel3/sgi/irix5/bar`
	Example 2: `foo/...` matches the *foo* directory itself, along with the entire directory tree under it.

SEE ALSO

cleartool subcommands: catcr, diffcr, find, findmerge, mkattr, mklabel
cc.magic, config_spec, makefile_ccase

NAME

xclearcase – primary ClearCase graphical interface utility

SYNOPSIS

xclearcase [**–fil·e** | **–att·ype** | **–brt·ype** | **–elt·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** | **–vtr·ee** [**–all**] [**–nme·rge**] [**–nco**] *pname …*]
[*X-options*]

DESCRIPTION

Invokes the ClearCase GUI (graphical user interface). For more information, see the *ClearCase User´s Manual*.

xclearcase is implemented as an X Windows application using a standard window system toolkit. See your X Windows documentation for a description of mouse and keyboard conventions.

OPTIONS AND ARGUMENTS

Selecting a Browser. *Default:* Starting *xclearcase* brings up the *main panel*, an enhanced file browser. (`xclearcase -file` has the same effect.)

–fil·e –att·ype –brt·ype –elt·ype –hlt·ype –lbt·ype –trt·ype –vtr·ee
Specifies a browser type. You can use exactly one of these options to specify the type of browser that appears when you invoke *xclearcase*. Note that `xclearcase -vtree` is equivalent to the *xlsvtree* and *cleartool xlsvtree* commands. See the *xlsvtree* manual page for details on starting a vtree browser.

X Windows Options. *Default:* None

X-options *xclearcase* accepts all the standard X Toolkit command-line options (for example, `–display` and `–geometry`), as described in the *X*(1) manual page. Quote the option string if it includes white space.

X RESOURCES

"Shell instance" names for the *xclearcase* browsers and transcript pad:

```
xclearcase.vtree
xclearcase.metatype
xclearcase.file
xclearcase.viewtag
xclearcase.vob
xclearcase.username
xclearcase.string
xclearcase.list
xclearcase.pool
xclearcase*transcript
```

EXAMPLES

- Start the ClearCase graphical user interface.

```
% xclearcase
```

SEE ALSO

X(1)
ClearCase User´s Manual

NAME xcleardiff – compare or merge text files graphically

SYNOPSIS

- Compare files:

  xcleardiff [**–tin·y**] [**–hst·ack** | **–vst·ack**] [*X-options*] *pname1 pname2* …

- Merge files:

  xcleardiff **–out** *output-pname* [**–f·orce**] [**–bas·e** *pname*] [**–tin·y**]
  [**–hst·ack** | **–vst·ack**]
  [**–qal·l**] [**–pause**] [*X-options*] *contrib-pname* …

DESCRIPTION

xcleardiff is a graphical diff and merge utility for text files. It implements the *xcompare* and *xmerge* methods for the predefined element types *text_file* and *compressed_text_file*. *xcleardiff* can also compare, but not merge, directory versions. On color display monitors, *xcleardiff* uses different colors to highlight changes, insertions, and deletions from one or more contributing files. During merge operations, input files are processed incrementally and, when necessary, interactively, to visibly construct a merged output file. You can edit the merged output as it is being built — either directly in the merged output display pane, or by invoking a separate text editor on the merged output — to add, delete, or change code manually, or to add comments.

xcleardiff is implemented as an X Windows application using a standard Motif toolkit. See your X Windows documentation for a description of general mouse and keyboard conventions.

INVOKING xcleardiff

You can invoke *xcleardiff* directly from the command line, specifying files or versions to compare or merge. However, because *xcleardiff* implements the *xcompare* and *xmerge* methods for the *text_file_delta* and *z_text_file_delta* type managers, the following *cleartool* subcommands, when applied to text files, also invoke *xcleardiff*:

- *xdiff*
- *xmerge*
- *findmerge* (with options `–xmerge` or `–okxmerge`)

The *xdiff*, *xmerge*, and *findmerge* commands include the advantage of some extra command options — optional ClearCase preprocessing — in the same way that *diff* and *merge* offer more flexibility than direct calls to the character-based *cleardiff* utility. See `xdiff -pred`, `xmerge -insert`, and `findmerge -ftag`, for example.

Invoke *xcleardiff* directly when you are working with text files that are not stored in a VOB.

Various buttons and menu options in the *xclearcase* graphical interface also invoke *xcleardiff*.

SETTING THE COLOR SCHEME

ClearCase GUI utilities support several predefined color schemes, which are collections of X resource settings. The schemes are stored in directory */usr/atria/config/ui/Schemes*, and include a special scheme for monochrome monitors called “Willis”.

A scheme may be specified in your standard X resources file (typically, *.Xdefaults* in your home directory). It takes the form:

```
*scheme: scheme_name
```

You can also use standard X Window System mechanisms to customize the *xcleardiff* window. The X class name is `xcleardiff`. The specific color-related resources are:

```
xcleardiff*promptBrightColor
xcleardiff*changeColor
xcleardiff*deleteColor
xcleardiff*insertColor
```

See also the *schemes* manual page.

OPTIONS AND ARGUMENTS

Font Size. *Default: xcleardiff* uses the font specified by the resource `xcleardiff*diffFont`.

–tin·y Uses a smaller font, in order to increase the amount of text displayed in each display pane.

Difference Pane Stacking. *Default:* Each of the two or more files being compared or merged is displayed in a separate subwindow, or *difference pane*. By default, these panes are displayed, or "stacked", horizontally (side by side), with the base contributor on the left.

–vst·ack Stacks the difference panes vertically, with the base contributor at the top.

–hst·ack Displays the difference panes horizontally (the default behavior).

Merged Output File. *Default:* For merge operations, you must specify a merged output file with the `–out` option. *xcleardiff* returns an error if the specified output file already exists (unless the output file is a checked-out version).

–f·orce (merge only) overwrite the merged output file, if it already exists.

–out *output-pname*
(merge only; required) Specifies the merged output file — a checked-out version or a standard operating system file.

Specifying a Base Contributor for a Merge Operation. *Default: xcleardiff* does not calculate a base contributor file (see *merge* for contrast). The first contributor file named on the command line becomes the base contributor, against which the one or more additional contributor files are compared. "Query on All" mode (`–qall`) is in effect by default but can be deactivated from the graphical interface.

–bas·e *pname*
(merge only) Makes *pname* the base contributor file for a merge. Using `–base` turns off "Query on All" mode, unless `–qall` is explicitly supplied. See also "Merge Automation".

Merge Automation. *Default:* If you do not specify a base contributor file with `–base`, "Query on All" mode is enabled automatically. In this mode, *xcleardiff* prompts you to accept or reject each change, insertion, or deletion found in contributor files 2 through *n* on the command line. The options described in this subsection have no effect.

If you *do* specify a base contributor file with –base, *xcleardiff* performs the merge automatically, pausing to prompt only if two or more contributor files modify the same section of the base contributor file. If all changes can be merged automatically, *xcleardiff* prompts you before saving the merged output file.

–qal·l (merge only) If a base contributor file is specified with –base, –qall enables "Query on All" mode. In this mode, *xcleardiff* prompts you to accept or reject each modification (relative to the base file) in each contributor file. You can toggle this mode interactively during the *xcleardiff* session.

–pause (merge only) If a base contributor file is specified with –base, –pause enables "Pause after Auto Decisions" mode. In this mode, *xcleardiff* pauses with the prompt Continue merge? after each automatic change. This gives you an opportunity to edit the merged output, or to abort the merge. You can toggle the pause mode interactively during the *xcleardiff* session.

X Windows Options. *Default:* None

X-options *xcleardiff* accepts all the standard X Toolkit command-line options (for example, –display and –geometry), as described in the *X*(1) manual page. Quote the option string if it includes white space.

Diff/Merge Contributor Files. *Default:* None. You must specify at least two files for a diff operation, and at least one file for a merge operation (two, if a base contributor file is not supplied with –base).

contrib-pname …
The files to be compared or merged. If a merge operation does not explicitly include a base contributor file with –base, the first *contrib-pname* becomes the base contributor file. For a diff operation, *xcleardiff* does not calculate a *common ancestor* (see *diff* for contrast); the first *contrib-pname* is the base file against which subsequent contributors are compared.

EXAMPLES

- Compare two files in different directories.

```
% xcleardiff test.c ~jpb/my_proj/test_NEW.c
```

- Use version-extended pathnames to compare a view-private file with two versions of a related file element.

```
% xcleardiff my_source_NEW.c my_source.c@@/main/LATEST my_source.c@@/main/4
```

SEE ALSO

cleartool commands: diff, xdiff, merge, xmerge
cleardiff, type_manager, schemes

NAME ptx_divider – separator page

DESCRIPTION

FOR POSITION ONLY

THIS PAGE TO BE REPLACED

BY A FULL-PAGE RUBYLITH AND

"Permuted Index"

FOR POSITION ONLY

BLANK PAGE WITH A

FULL-PAGE RUBYLITH

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-1613-030.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-965-0964
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389