U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 4646

National PDES Testbed
Report Series

A Guide to Configuration Management and the Revision Control System for Testbed Users

NATIONAL PDES TESTBED

NIST

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 4646

National PDES Testbed
Report Series

NATIONAL PDES TESTBED

A Guide to Configuration Management and the Revision Control System for Testbed Users

Scott Bodarky

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher,
Secretary of Commerce

National Institute of
Standards and Technology
John W. Lyons, Director

August 21, 1991

DEPARTMENT OF COMMERCE
UNITED STATES OF AMERICA

Contents

1 Introduction

Product Data Exchange using STEP (PDES) refers to the United States organizational activities in support of the development of the Standard for the Exchange of Product Model Data (STEP). These activities have resulted in the creation of large amounts of information and software, which reside in the PDES File System, a directory hierarchy called ~pdes on the computer system at the National Institute of Standards and Technology. The PDES software, documents, and data have been placed under configuration management [Ressler90] using the Revision Control System (RCS) [Tichy85], in order to ensure that change to these items occurs in a controlled manner, and that any services provided by them are done so as reliably as possible.

This document provides instructions for anyone needing to access this material. Enhancements to the RCS-based configuration management system will be documented in updates to this document and via electronic-mail (see Appendix A). A set of configuration management procedures will be published as another document.

The RCS-based system is intended to handle a variety of objects, including documents, software, and data. It is assumed that the reader has a basic familiarity with UNIX and an account on the systems at NIST. It is hoped that this system will be minimally encumbering while providing as stable and productive an environment as possible. This system is designed to make life easier for all users, from whom feedback is encouraged.

2 Document Conventions

There are sample command formats and code examples scattered throughout this document, all of which observe the following conventions:

- That which is to be typed verbatim by the user appears in **bold**.
- That which is to be typed by the user and is variable appears in ***boldface italics***.
- Verbatim RCS and Unix system messages appear in Helvetica.
- Variable RCS or Unix system messages appear in *Helvetica Oblique*.
- The Unix system prompt is the percent sign (%, in Helvetica), which will precede all Unix commands to be typed by the user, but is itself not to be typed.
- Items enclosed in square brackets [] are optional.

Note: Certain commercial products are identified in this paper in order to adequately specify configuration management procedures at NIST. This does not, however, imply recommendation or endorsement by NIST, nor does it imply that said products are necessarily the best available for their purpose.

The work described in this document was funded by the U.S. Government's Department of Defense Computer-aided Acquisition and Logistic Support (CALS) program and is not subject to copyright.

3 System Configuration

This section provides an overview of the configuration management environment and detailed instructions on setting up a local environment, for users needing to access items in ~pdes that are under configuration management.

3.1 The ~pdes File System

The principal repository for items under configuration management at the Testbed is a directory hierarchy called ~pdes. This hierarchy contains various entities used in support of the STEP development effort, including software tools, STEP test data, documents, and various other types of objects. The exact configuration of ~pdes and its contents tends to change periodically; these changes will be reflected by updates to this document and by electronic mail distribution (see Appendix A).

The following is a list of those sub-directories in ~pdes that are under configuration management by RCS. Each sub-directory may contain other sub-directories, which may themselves contain sub-directories, and so forth.

~pdes/data	contains STEP Testing data
~pdes/distribute	contains packaged software releases
~pdes/docs	contains documents
~pdes/generic.env	contains the Testbed generic environment
~pdes/include	contains header files for software tools
~pdes/man	contains the on-line man pages for software tools
~pdes/src	contains the source code for software tools
~pdes/bin	contains various executables
~pdes/arch/bin	contains architecture-specific executables

All manually generated files (i.e., those not output by some program) in these directories are managed by RCS, as are some binary executables.

Each directory contains the current, officially-released version of each of its files. Each of these files should be read-only, and should not be modified directly.

Each directory also contains a sub-directory called RCS, which contains an RCS archive for each controlled file. A user who wishes to modify a file must establish a symbolic link between his own account and the RCS directory, and then check out (into his account) a writable copy of the file from the appropriate archive. Instructions for doing this constitute the remainder of this section.

3.2 The Generic Configuration Management Set-up

A user's home directory must contain a sub-directory named pdes (~/pdes) that is structured like ~pdes. If the user wishes to access a particular file stored in the ~pdes hierarchy, his own pdes hierarchy must contain copies of all the intermediate nodes. The user's copy of the file will have the same relative path as the copy in ~pdes, differing only by the inclusion/exclusion of the slash between the tilde and the pdes.

Once the relevant portion of the pdes hierarchy has been duplicated, the user must establish a symbolic link from his copy of the file's source directory to the RCS sub-directory in the ~pdes source directory. For example, if a user wanted to modify the contents of a fictitious directory called ~pdes/stuff, he would create the configuration illustrated in Figure 3.1, by typing the following commands (from his home directory):

% mkdir pdes{,/stuff}
% ln -s ~pdes/stuff/RCS ~/pdes/stuff/RCS

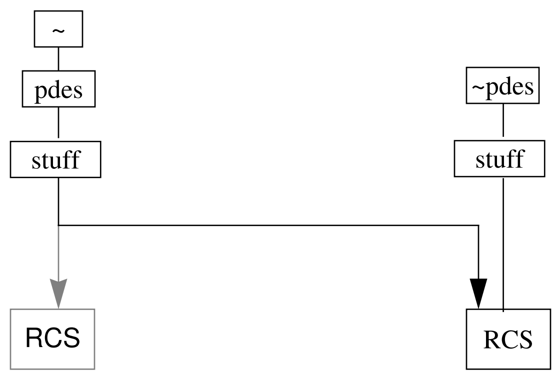

Figure 3.1 The RCS File-System Configuration

Once the source directory has been created, the user can check out a copy of the file via RCS (see Section 5), which extracts a copy of the file from its archive in ~pdes and deposits it in the user's directory. In the above example, the file would go into ~/pdes/stuff. Any modifications (or testing, in the case of software) to the file occur in the user's account. After the desired modifications are made, the file is checked back in as a new revision (see Section 6).

Important: It is critical that each user's path contain ~/pdes/bin before the /depot directories.

3.3 Additional Notes for Software Developers

The guidelines and procedures that will govern development of software under configuration management are being drafted as part of the recently launched Testbed Readiness Program. They will be published as two separate documents, called "PDES Software Development Guidelines" and "PDES Software Configuration Management Procedures". The following comments are important, however, and are worth mentioning here.

The software tools under configuration management reside in ~pdes/src and ~pdes/include. The actual files that compose the tools, and that reside in these directories support dependencies in other files and are used to build releases; they are write-only, and should never be modified in the course of development. Each file also has a designated release state, which will be one of the following: **Exp** (experimental), **Alpha**, **Beta**, or **Stable**.

In addition to the files and the RCS sub-directory, each source directory will contain a shell script called CheckOut, which automatically checks out a read-only copy of the officially-released version of each file in that directory. The CheckOut script checks out these files according to their release state (see Section 6.5), and therefore requires updating only when a new file is added or an existing file changes state (e.g., is promoted from **alpha** to **beta**). Here is an example CheckOut script:

```
#!/bin/sh
co -salpha Makefile
co -sbeta hash.c hash.h
co -salpha class.c class.h dictionary.c dictionary.h \
    error.c error.h linked_list.c linked_list.h
```

If the changes to a given file constitute a new release, but the state of the file has not changed, the developer should move to ~pdes/src/stuff and run the CheckOut script. If the state of the file has changed, the CheckOut script should be modified to reflect this and then run from the appropriate directory in ~pdes.

3.4 File Conventions

RCS has an automatic keyword-expansion facility (see Section 4.5), and each text file in the ~pdes hierarchy should contain the RCS reserved-word **Id**. Files containing C code should contain the string **static char rcsid[] = "Id";**, so that it will be possible to easily determine from which version of source code a binary was generated. C header files need only contain the string /***Id*/**. Shell scripts should contain the string **#Id**. Documents produced with various publishing systems (e.g., Framemaker or WordPerfect) may also take advantage of this mechanism, although it is not required. It is important that the **Id** be in a comment, so that it is visible on the screen, but not in the printed document.

A revision of a source file that is to be included in a software release should be assigned a symbolic name (see Section 6.4) indicative of its status, with the hierarchical levels of the revision number separated by hyphens (e.g., bpr2-1). This will enable check-outs predicated on the symbolic name. Note that periods are not permitted in symbolic names.

4 RCS Archives

A file configured under RCS is maintained in an archive, which has the same name as the file, with a **,v** appended. An archive is an ASCII text file which contains the revision history and a complete description of the file. Each version of the file, including the original, is called a revision. One revision is defined to be the current revision, which means that operations that do not specify a specific revision will be applied to it. A physical copy of a revision, called a working-file, can be checked out from an archive. Each revision (except the current revision—see section 4.4) is stored inside an archive as a set of changes relative to another revision, called a delta. An archive contains, in order, an administrative node, the revision tree (implemented as a linked list), a general description of the archive, and the deltas. The remainder of this section describes the contents of an RCS archive. Those readers who are looking to get started with development as quickly as possible can skip ahead to Section 5, and refer back to this section as time or need permits.

The next several sections outline the exact syntax of an RCS archive. Although it is perhaps outside the scope of a basic users guide, it has been included for two reasons. First, knowledge of what is contained in archives serves to make the functioning of RCS more intuitive. Instead of a bevy of commands with multifarious options, RCS can be seen as maintaining a data structure whose slots can be manipulated in various ways. Also, the information is organized differently from other available documentation, and some of it is not available in the RCS documentation.

4.1 The Administrative Node

The first item in an archive is its administrative node, the syntax of which follows:

head [*head-num*];
branch [*branch-num*];
access [*access-list*];
locks [*lock-list*];
symbols [*symbol-list*];
comment [@*string*@];

where

head-num	is the revision number of the current revision.
branch-num	is the revision number of the root of the default branch of the revision tree, which means that operations on entire branches (as opposed to specific revisions) that do not specify their targets will affect the branch whose root is *branch-num*; if there is no *branch-num*, the highest branch on the trunk (see section 4.2) is the default.
access-list	is the access list for the archive. Users mentioned in the list are granted write access to revisions in the archive. If the list is empty, access is unrestricted.
lock-list	is a list of security-status determinants. The lock list for each archive will always contain the word strict, which indicates that strict locking is engaged. Strict locking is a security feature which prevents more than one writable working-file from being checked out at a time. Strict locking can be disengaged (see the on-line manual), in which case strict will disappear from the locks list. When a writable working-file is checked out under strict locking, the name of the user who checked it out is added to the lock list. The name is removed when the revision is checked back in.
symbols	is a list of symbolic names (if any), as defined by **ci -n**.
comment	is a space reserved for comments.

4.2 The Revision Tree

The revision tree consists of a sequence of nodes, each of which represents a delta and contains a link to its parent. Each node is of the following form:

rev-num
date *date-string*; author *author-name*; state *state-name*;
branches [*branch-list*];
next [*next-rev*];

where

rev-num	is the revision number of the node.
date-string	is the date the revision was checked in.
author	is the user who checked in the revision.
state-name	indicates what state the archive is in. The set of possible states is user-definable. Currently, RCS is configured to recognize the following states: Exp (Experimental), Alpha, Beta, and Stable. All newly created archives and newly checked-in revisions are designated Exp automatically, unless another state is explicitly assigned (see sections 6.5 and 7.6).
branch-list	is a list of the revision numbers that are at the top of any side-branches of the revision tree that issue from the node.
next-rev	is the revision number of the node's parent in the revision tree.

4.3 The General Description

The general description of the archive is entered at check-in time, though it can be modified later (see section 7.7). It appears as follows:

desc
@[*description*]@

where *description* is a textual description which may or may not span more than one line. The @ character is the RCS string delimiter.

4.4 The Deltas

The current revision is stored in its entirety. Each other revision is stored as a delta, which consists of a list of instructions for undoing the changes that were made to the revision. When a revision is checked out, RCS constructs it by starting at the current-revision node and traversing the path in the revision tree to the node of desired revision. If RCS is ascending the tree it follows the instructions in the deltas, which undo changes. If it is descending or moving laterally in the tree it reverses the instructions in the deltas, thus making changes (see Figure 4.1). A delta is formatted as follows:

```
rev-num
log
@[description]
text
@[content]
@
```

where

rev-num	is the revision number
description	is a description of the revision. For the initial revision, it is automatically set to Initial Revision; for subsequent revisions the user supplies the description at check-in.
content	is either the complete text of the revision (if the revision is the current one) or the instructions on how to construct the revision from the node that is one closer to the current revision (see figure Figure 4.1). Instructions are of the form: *xr s* [t], where *x* is either a (for add) or d (for delete), *r* is the first line to be affected, and *s* is the number of lines to be affected. If *x* is a, the optional argument *t* is included, containing the text to be added.

Note the following example:

```
1.0
log
@ Initial Revision
text
@ d2 2
a0 1
This is line 1
a2 1
This is line 3
@

1.1
log
@ Lines 1 and 3 removed
text
@This is line 2
This is line 4
This is line 5
@
```

We can see that revision 1.1 reads:

This is line 2
This is line 4
This is line 5

Were we to check out revision 1.0, RCS would build it from revision 1.1 according to the instructions in delta 1.0. First RCS would delete the second and third lines (d2 2). Then it would add one line at position 0 (a0 1 This is line 1) and one more at position 2 (a2 1 This is line 3) to give us our initial revision. Note that subsequent instructions do not build on each other, but refer back to the unrestored revision. Our result:

This is line 1
This is line 2
This is line 3

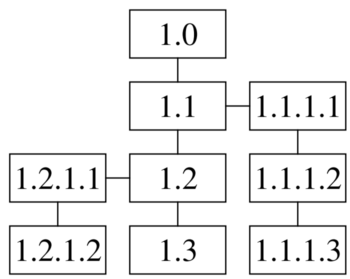

Figure 4.1: Revision 1.0 would be reconstructed by modifying the text of revision 1.3 with the instructions in deltas 1.2, 1.1, and 1.0, in that order. Revision 1.1.1.3, however, would be obtained by modifying the text of revision 1.3 according to the instructions in deltas 1.2, 1.1, 1.1.1.1, 1.1.1.2, and 1.1.1.3, also in order.

4.5 RCS Reserved Keywords

There are certain reserved keyword clauses that can be included in a revision. When a revision containing them is checked in, the clauses are expanded to include values which reflect conditions at that particular moment. When next checked out, the values from the previous check-in will appear in the clauses, but will be replaced by new values when checked in again. These clauses are useful for maintaining topical information. Each keyword is entered into a revision enclosed in dollar signs, as follows:

$*keyword*$

When the revision containing one or more such keyword clauses is checked in, values are assigned, and the clauses appear as follows:

$*keyword*:*value*$

For example, if **$Date$** were included in a revision which was checked in on September 4, 1990 at 2:52:51 PM, the revision would contain the following string:

```
$Date: 90/09/04 14:52:51 $
```

Following is a list of all the permissible keywords and brief descriptions of each. The keywords are case-sensitive.

Author	user who checked in the revision
Date	date and time of check-in
Locker	user who currently has write-access, if any
Log	log message from last check-in
RCSfile	name of the archive
Revision	revision number
Source	name of the archive, including path
State	state of the revision
Header	set of Source, Revision, Date, Author, State, and Locker
Id	set of RCSfile, Revision, Date, Author, State, and Locker

The **ident** command lists all keyword clauses in a file:

%**ident** ***filename***

5 Checking Out Files From Archives

The RCS check-out command (**co**) is used to extract a file from its RCS archive. This section provides instruction on important functions one can perform with this command:

- viewing a read-only copy of a file
- accessing a writable copy of a file
- checking out by symbolic name
- checking out by date or time
-checking out by state

For additional information, consult the on-line man pages.

5.1 Checking Out with Read Access Only

To view a file, a check-out (**co**) command must be issued:

%**co** ***filename***

Either the name of the file or the name of the archive (i.e. the **,v** is optional) may be supplied. RCS will then deposit a read-only working-file in the user's development directory, where it may be looked at or deleted. Any number of read-only working-files may be checked out at a time to any number of users.

5.2 Checking Out with Write Access

To obtain a writable working-file, the check-out command with the **-l** option must be used:

%co -l ***filename***

The -l stands for lock, and a co command which includes it locks the archive so that no other users can check out writable copies of the file. The lock is removed when the writable copy is checked back in (see section 6). Note that a file cannot be checked in if the archive is locked, so if a writable file is checked out and then accidentally deleted, the lock is left in place. To remove such a lock from the archive, type:

%rcs -u ***filename***

If a user attempts to check out a writable copy of a revision that another user has locked, RCS will not allow the check out, but will supply the username of the holder of the lock. The command **rcs -u** can be used to break a lock imposed by someone else. As this somewhat defeats the purpose of using RCS in the first place, it is not a recommended practice, and should only be done if absolutely necessary. Injudicious use of **rcs -u** can result in the locks getting out of sync, a circumstance that is most undesirable. When a lock is broken, automatic notification is mailed to the holder of the lock.

A single user can have any number of files checked out and locked simultaneously, and can also perform repeated check-outs on a single file (without checking it in). Each checked-out revision is automatically placed in the development directory, and will replace its predecessor there. If the file to be replaced has been modified, RCS will ask for verification before deleting it. Care should be exercised, because supplying such verification will erase any changes that have been made.

Should a file be inadvertently checked out, it can simply be checked back in. RCS will recognize that no changes have been made and ask Are you sure?. If the answer is **no**, RCS will delete the working-file and release the lock.

5.3 Checking Out by Revision or Symbolic Name

In an RCS archive, one revision (usually the most recent one) is defined as the current revision, and is checked out by default. The -r option with the desired revision number is used to check out a revision other than the current one:

%co -l -r1.2 some_file.c

The -r option can also be used to check out a working file based on a symbolic name, as assigned with ci -n (see section 6.4):

%co -r*symbolic_name filename*

5.4 Checking Out by Date or Time

A revision can be checked out according to the date and/or time it was checked in, with the **-d** option. RCS allows considerable variation in the formatting of the **-d** option, but it is recommended that users use the following syntax:

%co -d"15:34 August 20 1990" some_file.c

The date-string must be in quotations if it contains spaces. It can contain one or more of the eight fields described below, in order of precedence:

year: 2- or 4-digit string (e.g. 1990 or 90)
month: the full name or any sufficiently distinct abbreviation thereof, either with or without a period (e.g. February, Feb., Feb, Fe--an F alone would not suffice, as it could be confused with Friday)
day: either the number (with respect to the month) or the name (subject to the same syntactical requirements as the month)
hour: 12-hour notation or military time
minute: number
seconds: number
am/pm: optional
time zone: any (e.g. CDT, EST, PDT)

The fields can be rearranged or omitted. When fields are omitted, those of higher precedence than the highest supplied are assumed to be current, and those of lower precedence to have the lowest value possible. For example, if a particular revision is desired, but the user doesn't recall its number, the date may be used to retrieve it. If it is known that the revision is, say, the first one done in August, the following would work:

%co -dAugust some_file.c

The year field is of higher precedence than the month field, so the current year is assumed. The remaining fields have lower precedence and assume their minimum valid values.

5.5 Checking Out by State

It is possible to assign states to specific revisions (see sections 6.5 and 7.6). The four states that RCS is configured to deal with are Exp (experimental), Alpha, Beta, and Stable. To obtain a revision based on state, use the **-s** option:

%co -sBeta some_file.c

If more than one revision has the same state, the most recent one will be chosen.

6 Checking Files Into RCS

The RCS check-in command (ci) is used to add a new set of changes to an RCS archive. This section provides instruction on the important functions one can perform with this command:

- checking in changes
- explicitly assigning a revision number
- assigning a date
- assigning a symbolic name
- assigning a state

For additional information, consult the on-line man pages.

6.1 Checking In

After all desired modifications have been made, the file must be checked back into the archive. Issuing a check-in command with no options will increment the level number and create a new revision. If revision 1.3 of some_file.c is checked out and modified, checking in as follows will result in revision 1.4:

```
%ci some_file.c
RCS/test.c,v <-- test.c
new revision: 1.4; previous revision: 1.3
enter log message:
(terminate with ^D or single '.')
>>
```

As shown, RCS will confirm the check-in and prompt for comments. If a user wishes to enter comments, they must be typed at the >> prompt. There is no limit to the number of lines; the log entry is terminated with either a **^D** or a period alone on a line.

6.2 Assigning a Revision Number

Revision numbers define the logical relationships between revisions. Generally, the initial version of a file entered into a configuration management system (CMS) is called a baseline. If the baseline is a release as defined in the next paragraph, it should be assigned a revision number of 1.0. If it is not a release, it should be assigned a revision number of 0.0. The first digit of a revision number is termed the release number. The second and subsequent digits are called revision numbers.

A release is that incarnation of a series of versions of an item under configuration management that is sufficiently changed from its predecessor such that it can be defined as a new conceptual entity. If the incarnation is the oldest ancestor of the series, it is the baseline, and is of course different from its predecessor, the empty set. The revision number of a release is generally of the form ***x*.0**, where ***x*** is the release number. Having zero as the revision number reflects that no changes have yet been made to the release.

The revision numbers order the lower-level changes that result in the higher-level ones which differentiate releases. The revision number for a new release is automatically set to zero and incremented by one as each subsequent revision is checked in.

RCS permits assignment of a specific revision number to a revision, via the **-r** option:

%ci -r2.0 some_file.c

6.3 Assigning a Date

The **-d** option is used to specify the date:

%ci -d"August 30" some_file.c

6.4 Assigning a Symbolic Name

A symbolic name is a string that can be assigned to a specific revision. The string becomes functionally equivalent to the revision number. The symbolic name can consist of any alphanumeric characters except periods. The assignment is done with the **-n** option:

%ci -ndistribution some_file.c

For instructions on removal of a symbolic name, see section 7.4.

6.5 Assigning a State

Every revision has a state. If none is explicitly assigned, the revision is automatically designated experimental (**exp**). Other states that RCS recognizes are **alpha**, **beta**, and **stable**. When a file is designated an **alpha** release (cf. the forthcoming Procedures Document mentioned in the introduction), an entry is made in the CheckOut script in the appropriate directory in ~pdes. The commands in CheckOut should reference a file by state, and should therefore only require modification when the file changes state (i.e., from **alpha** to **beta** or from **beta** to **stable**). If the changed file is a component in a binary or library, the developer should rebuild the binary/library and deposit it in ~pdes/arch/bin or ~pdes/arch/lib, as appropriate. The developer should be sure to delete all object files after building a library or binary.

A state is assigned with the -s option:

%ci -sAlpha some_file.c

7 The RCS Command

The **rcs** command is used for a variety of functions, most of them administrative in nature. Some of the functions of **rcs** are identical to those performed by **ci**, but provide the opportunity to change archive attributes without going through a check-out and subsequent check-in.

7.1 Manipulating Access Lists

Each archive has an access list, though no restrictions are enabled so long as the list is empty. The **-a** option is used to add users to the access list of an archive:

%rcs -a*user_names filename*

Names on the list should be separated by commas. The **-A** option is used to duplicate the access list of one archive for another, as follows:

%rcs -A*file1 file2*

where ***file1*** is the source and ***file2*** the destination. Use the **-e** option to remove names from an access list:

%rcs -e[***user_names***] ***filename***

If no user-names are supplied, the entire access list is deleted. This makes the file available to everyone (subject, of course, to UNIX security measures).

7.2 Designating the Default Branch

One branch of the revision tree is considered the default branch. Any RCS operations will be directed at this branch, if none other is indicated. The default branch can be designated as follows:

%rcs -b[***rev***] ***filename***

The revision number supplied should be that of the top node on the desired branch; if no revision number is supplied, the highest branch on the trunk is designated the default.

7.3 Locking an Archive

Although the standard way to lock an archive is to use **co -l**, there may be times when it is desirable to prevent anyone from modifying a file. One easy way to do this is to lock the archive as follows:

%rcs -l[***rev***] ***filename***

To unlock the archive:

%**rcs -u**[*rev*] ***filename***

7.4 Symbolic Names

To assign a symbolic name to a revision, use either **ci -n** (see section 6.4) or **rcs -n**:

%**rcs -n*symbolic_name filename***

To remove a symbolic name, use **-n** with no argument:

%**rcs -n *filename***

To replace a previously assigned symbolic name with a new one, use **-N**:

%**rcs -N*new_symbolic_name filename***

7.5 Updating Archives

To delete old and obsolete revisions from an archive, use:

%**rcs -o*range filename***

where ***range*** is of the form [***rev1***][**-**][***rev2***]. One revision number alone will result in the deletion of that revision. A revision number with a trailing hyphen will result in the deletion of that revision and all that follow it on its branch. A revision number with a leading hyphen will result in the deletion of that revision and all that precede it on its branch.

This procedure is recommended when appropriate. It frees up physical storage space and results in cleaner and more efficient archives.

7.6 Assigning States

A revision can be assigned a state with ci -s (see section 6.5) or as follows:

%**rcs -s*state*[:*rev*] *filename***

If no revision is specified, the current one is assumed.

7.7 Modifying the General Description

Each archive has a general description which describes the entity under configuration management, as opposed to comments on specific revisions. To modify the general description, use the **-t** option as follows:

%rcs -t[*txtfile*] ***filename***

The descriptive text in ***txtfile*** will be inserted into the archive. If the argument is left off, RCS will prompt the user to input the text through the standard input. When finished, terminate the text with a **^D** alone on a line.

8 Getting Information About Archives (rlog)

The rlog command will generate a listing of information from the archive of a particular file:

%rlog smush.c

```
RCS file: RCS/smush.c,v
Working file: smush
head: 1.2
branch:
locks: strict
        bodarky: 1.2
access list:
symbolic names:
comment leader: "# "
keyword substitution: kv
total revisions: 2;     selected revisions: 2
description:
This file is an ersatz application.
----------------------------
revision 1.2    locked by: bodarky
date: 1991/03/04 15:27:18;  author: bodarky;  state: Exp;  lines: +10 -11
fixed post-impact re-inflation bug
----------------------------
revision 1.1
date: 1991/02/28 18:39:29;  author: bodarky;  state: Exp;
Initial revision
 =============================================================
```

9 RCS and other Gnu Software

RCS is distributed by the Free Software Foundation[1], which also distributes other software tools that can interact with RCS. Two of these are discussed in the following sections.

9.1 Gnumake

1 Free Software Foundation
675 Mass Ave.
Cambridge, MA. 02139
USA

Make is a generic UNIX utility which can be used to automate the installation of large software systems. The Gnu version of this tool is called gnumake. No provision for RCS need be made in a Makefile to be used with gnumake; if a file is referenced in such a Makefile and not found, gnumake will search for an appropriately named RCS archive and, if it finds it, check out the file automatically.

9.2 Gnu Emacs

Gnu also has its own version of the powerful text editor emacs, called Gnu emacs. Emacs can be very intimidating and confusing for the novice, and it is recommended that you not attempt the following instructions unless you are already familiar with it.

When emacs is first invoked, it searches the user's home directory for a file called .emacs, which contains user-specific set-up information. The following three lines need to be somewhere in the file in order for Gnu emacs to properly handle RCS:

(setq find-file-not-found-hooks '(my-RCS-file))
(autoload 'ci "rcs" "RCS ci/co mode" t)
(autoload 'my-RCS-file "rcs" "rcs ci/co mode" t)

When you have an emacs session running, and try to access a file, emacs will generally load it from the current directory. If the file is not there, emacs will open a new-file buffer with the name you have provided. When RCS-enabled, emacs will try to locate an appropriately named RCS archive before it opens a new buffer. If the archive is found, gnu-emacs will check out the file automatically, after asking you whether you want a read-only or writable copy.

Emacs does not automate check-in. After your modifications are complete, you must issue an explicit check-in command to register your work in the archive. First type

<Esc>x ci

by holding down the Escape key and pressing lower case **x**, then releasing both and typing **ci**, and then pressing the return key. You will then be prompted to enter a log message. Do so if you choose, and then type:

^c ^c

by holding down the Control key and pressing lower case **c** twice. The file will then be checked into the archive at this point.

10 Glossary

Archive: A file containing administrative information for some entity configured under RCS, the text of said entity, and a set of revision instructions (deltas) detailing all the modifications that have been made to the entity. An RCS archive can be recognized by a ‘,v’ appended to the file-name.

Current Revision: The current revision is the default target for all operations that do not specify a revision. It is the only revision in an archive for which the entire text is stored. Note: the current revision is not necessarily the most recent revision, although it almost always is.

Delta: A complete description of the differences between one revision of a file and its predecessor in the revision history.

Initial Revision: The first revision of a file, usually the text that is first entered into an **RCS** archive. Its revision number is generally 1.0.

Level Number: The second (and any subsequent) field of a revision number.

PDES File System: A dedicated file system (called ~pdes) containing various items which support PDES. Items stored there include software, documents, test data, etc.

Release: A release is a conceptual entity that consists of one or more related software items. In most cases, only one version of each item is included in the release.

Release Number: The first field of a revision number.

Revision: A snapshot in time of a file configured under RCS and under development. A new revision is created when a modified version of a file is checked into an RCS archive. A revision is denoted by its revision number.

Revision Number: The identifying number of a revision, establishing its place in the revision history.

Revision History: The set of all revisions, each of which has a distinct revision number. The revision numbers are arranged hierarchically, and the release and/or level numbers are incremented with each subsequent revision.

Revision Tree: The data structure in which the revision history of an archive is stored. The root node of the tree contains administrative information. It has one child (the sub-root, if you will), which is the initial version of the file. Subsequent revisions are stored below the sub-root, either in the tree’s trunk or in side branches (see Figure 10.1):

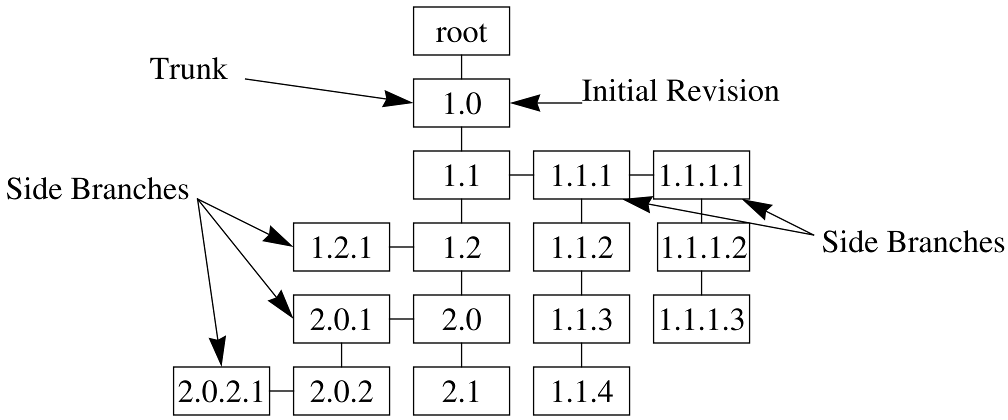

Figure 10.1: A typical revision tree

Side Branch (of a revision tree): A side branch contains revisions with multiple level numbers (e.g. 1.1.1, 1.1.2, 1.1.2.1, 1.1.2.2, etc.).

Trunk (of a revision tree): The trunk consists of those revisions with a single level number (e.g. 1.1, 1.2, 1.3, 1.4, 2.0, 2.1, etc.), which constitute the main path of revision.

Working File: A file that has been checked out of an RCS archive.

11 References

[Clark90] Clark, Stephen Nowland, An Introduction to the NIST PDES Toolkit, NISTIR 4336, The National Institute of Standards and Technology, MD, May 1990.

[McLean91] McLean, Chuck, National PDES Testbed Readiness Program: Policies and Procedures, Internal Memorandum, March 1991.

[Ressler90] Ressler, Sanford, and Katz, Susan, Development Plan: Configuration Management Systems and Services, NISTIR 4413, The National Institute of Standards and Technology, MD, September 1990.

[Tichy85] Tichy, Walter F., RCS – A System for Version Control, Purdue University, Indiana, July 1985.

Appendix A The Electronic Mailing Lists/Newsgroups

In order to improve communication between users of the configuration management system and to facilitate dissemination of information concerning it, two local Usenet newsgroups have been established at NIST. These newsgroups are also available as electronic mailing lists for users off-site or anyone who would rather receive mail than read news. The two lists are pdes.config.discussion and pdes.config.policy. To be added to (or removed from) either mailing list, send an e-mail message containing your request to the NIST Configuration Management System Administrator at pdescmsa@cme.nist.gov.

The discussion newsgroup is an open forum for general discussion of (or questions about) the configuration management system. To submit comments or questions to the list, enclose them in an e-mail message addressed to pdes-config.cme.nist.gov. Your message will appear in the local newsgroup and be distributed as a mail message to everyone on the list, as will responses to your message.

The policy newsgroup is a read-only newsgroup for quick dissemination of information, new policy, and changes to old policy. Only the NIST Configuration Management System Administrator can send messages to this list; messages from anyone else will be re-routed to the Administrator.