

# PRCS: The Project Revision Control System

Josh MacDonald<sup>1</sup>, Paul N. Hilfinger<sup>1</sup>, and Luigi Semenzato<sup>2</sup>

<sup>1</sup> University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, Berkeley CA 94720, USA,

<sup>2</sup> National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, Berkeley CA 94720, USA

**Abstract.** PRCS is an attempt to provide a version-control system for collections of files with a simple operational model, a clean user interface, and high performance. PRCS is characterized by the use of project description files to input most commands, instead of a point-and-click or a line-oriented interface. It employs optimistic concurrency control and encourages operations on the entire project rather than individual files. Although its current implementation uses RCS in the back-end, the interface completely hides its presence. PRCS is free. This paper describes the advantages and disadvantages of our approach, and discusses implementation issues.

## 1 Overview

PRCS is an attempt at producing a version control system for collection of files that is competitive with existing commercial and free systems with regard to ease of use, implementation, and maintenance. PRCS borrows freely from well-established concepts and ideas in this area, but re-engineers them into a small set of orthogonal features, a clean version model, and a simple yet powerful user interface based on text editing.

The design and development of PRCS was partly motivated by the authors' dissatisfaction with existing systems. Commercial systems tend to be large, feature-laden, and expensive. They often have fancy graphical user interfaces, which can make many operations intuitive, but typically present a sharp threshold beyond which even conceptually simple operations become exceedingly cumbersome. CVS (Concurrent Version System)[1], the standard non-commercial solution, is widely used but we find that many aspects of its operation, administration, and user interface are unnecessarily complex, mostly because of its choice of version model and its explicit dependency on RCS.

CVS and several other systems are designed around the idea that version control for a group of files can be achieved by grouping together many single-file version-control operations. PRCS defines a project version as a labeled snapshot of a group of files, and provides operations on project versions as a whole. Our experience shows that focusing on this model results in a cleaner system than a model cast explicitly as groups of single-file operations.

One notable difference between PRCS and other systems is its user interface. In the PRCS model, each version of a group of files contains one distinguished file,

called the version descriptor. This file contains a description of the files included in that particular version. All user-controlled project settings are entered and manipulated through this file. Indeed, many user-interface issues become text-editing problems from the start, rather than requiring a graphical user interface or numerous command-line utilities for maintaining project state.

Additionally, PRCS includes several novel features, including an improved keyword replacement mechanism and a flexible way of incorporating PRCS into external programs.

In the remainder of this paper, we provide a small example (§3), and discuss the system design and operational model (§2), related work (§4), branch control (§5), distinguishing features (§6), implementation considerations and implications for back-end storage management (§7), and future work (§8).

## 2 Operational Model

PRCS presents the user with the abstraction of named *projects*, which are collections of *project versions* (or simply *versions*), each of which is a snapshot of a set of files arranged into a directory tree. Every project version is labeled with a *version name*, which is unique within that version's project. Finally, a PRCS *repository* contains a group of projects.

Each version contains one distinguished file in its top-level directory, known as its *project-version descriptor*, or simply *version descriptor*. This is an ordinary text file that identifies the version and contains a list of the version's constituent files. When a file is added to a project version for the first time, it is assigned a unique, internal identifier, which follows the file through its history, even as it is renamed. This identifier is known as the *internal file family*. The version descriptor contains a mapping between internal file families and file names. We give more detail in §6.1.

Users modify directory structures of files independently of the repository. We use the adjective *working* to distinguish these files and directories from those that are stored in the repository: thus *working version*, *working file*, and *working version descriptor*. When we say that a project version is a snapshot of a directory structure, we typically mean a snapshot of such a working project version.

Before any working version is *checked in* (the usual term for “taking a snapshot”) PRCS modifies its descriptor to reflect the new (repository) version. In particular, the resulting version descriptors contain a record of the identity of the versions from which they were derived, thus inducing a partial order on versions that serves to define ancestry.

Within this framework, PRCS provides the following major operations:

**Checkin** deposits a labeled snapshot of a working version in the repository.  
**Checkout** reconstructs any project version, identified by project and version label.

**Diff** compares project versions.

**Merge** reconciles two project versions interactively by modifying the working project version. PRCS uses the notion of ancestry described above to