Migration of the Gaudi and LHCb software repositories from CVS to Subversion

M Clemencic¹ and H Degaudenzi² on behalf of the LHCb Collaboration
¹ CERN, CH-1211 Genève 23, Switzerland
² EPFL, CH-1015 Lausanne, Switzerland
E-mail: marco.clemencic@cern.ch, hubert.degaudenzi@cern.ch

Abstract. A common code repository is of primary importance in a distributed development environment such as large HEP experiments. CVS (Concurrent Versions System) has been used in the past years at CERN for the hosting of shared software repositories, among which were the repositories for the Gaudi Framework and the LHCb software projects.

Many developers around the world produced alternative systems to share code and revisions among several developers, mainly to overcome the limitations in CVS, and CERN has recently started a new service for code hosting based on the version control system Subversion.

The differences between CVS and Subversion and the way the code was organized in Gaudi and LHCb CVS repositories required careful study and planning of the migration. Special care was used to define the organization of the new Subversion repository. To avoid as much as possible disruption in the development cycle, the migration has been gradual with the help of tools developed explicitly to hide the differences between the two systems.

The principles guiding the migration steps, the organization of the Subversion repository and the tools developed will be presented, as well as the problems encountered both from the librarian and the user points of view.

1. Introduction
The aim of this paper is to present the migration of the LHCb software repositories from the CVS to the Subversion version control system with a particular focus on the intrinsic problems of this kind of migration and the way they have been addressed.

1.1. Version control systems
The terms version control system identify a category of tools that allow their users to keep track of the evolution of files. They are mainly used for software source code or more in general for text files, but some of them can efficiently keep track of binary files (like images).

The main features of a version control system are:
- tracking versions (or revisions) of files, i.e. adding, retrieving and comparing different versions of the files
- flagging and identifying special versions of the files with mnemonic names (tags)

Given the basic functionalities, the various version control systems differ by the additional features they provide (e.g. tracking of copies of files) and the working model (e.g. centralized vs. distributed).
1.1.1. CVS  The Concurrent Versions System, best known as CVS[1], is one of the best known version control systems. Many software projects have been hosted on CVS repositories since its birth in the late 80s, and several projects still use it despite the lack of active development and its limitations, as, for example, the expensive tag and branch operations, the inability to track copies and renames of files or directories and the lack of atomic operations.

1.1.2. Subversion  The version control system Subversion[2] (also known as SVN) was created in the year 2000 with the explicit aim of replacing CVS by providing a similar (or equivalent) user interface and fixing some of its limitations.

Subversion, while very mature, is still being actively developed.

1.2. Why to migrate

The choice to migrate the LHCb software repository from CVS to Subversion has a technical component and a practical one.

On the technical side, a Subversion repository allows a better control of what happens in the repository than does CVS. For example, Subversion gives the possibility to restrict the read access to some entries only to a group of authenticated users (it was not possible with CVS). Other advantages of SVN, with respect to CVS, are the possibility to keep track of the copies of files and directories and the so called atomic commits, which prevent that only part of the desired changes are stored in the repository if something fails during the commit. On the other hand, the particular design of the SVN repository interface (versioned filesystem) and the precise choice of not adding a built-in concept of tags and branches give the Subversion repository an extreme flexibility, at the price of being difficult to understand and adapt to for users with a CVS background.

On the practical side, in 2008, CERN IT, given the high demand for a Subversion service, decided to schedule a retirement of the CVS service (first reducing the support to a best-effort level, then stop it on a date agreed with the experiments) and start providing a Subversion service.

2. Preparing the migration

2.1. The structure

As already briefly mentioned, Subversion features a repository interface of a versioned filesystem (i.e. it looks like a hierarchy of directories and files and it remembers all the changes that occurred in the hierarchy) and doesn’t have a built-in concept of tags and branches. These design choices are correlated, because if the repository can remember all the operation done on the hierarchy of directories, tags and branches can be emulated with copies, in the sense that a copy of a directory at a given moment in the history of the repository can be interpreted as a tag (or a branch).

The SVN documentation suggests to organize the repository with one directory called trunk for the main development line of the project, one directory called tags to host copies of the versions of the trunk directory that the developer want to label with mnemonic names (equivalent to CVS tags) and one directory called branches for the copies of the versions of the trunk directory from which to start some parallel development. Of course, the copies in tags can come from the subdirectories in branches and vice-versa.

In Subversion, the trunk, tags and branches names are not enforced, but just suggested as best practice and are a de-facto standard. Users can organize the repository the way they see best, but it is mandatory that the structure and policies are defined and followed carefully, to avoid problems and inconsistencies. It has to be noted, anyway, that the versioned filesystem concept makes it easy to track and undo any mistake easily.
To define the structure of the LHCb repository, we had to take into account the special needs of the management of our software. In particular it is fundamental to define the granularity of the tags, because in SVN the tags are emulated with copies of directories.

The LHCb software is organized in packages (minimal units that can be versioned) and projects (coherent groups of packages deployed together). We have the need to be able to check out and tag packages, but also, in particular cases, to check out a whole project. To cover these two use cases, the repository has been structured with a first level corresponding to the software projects (containers of packages). Immediately below the project level, we have the three standard directories trunk, tags and branches. The trunk of a project contains one directory for each package belonging to the project plus the special directory (cmt) required by the build tool we use (CMT[3, 4]), so that the content of the trunk is identical to the layout of the project on disc. The content of the tags directory (and similarly of the branches one) features again one directory per package and each of these directories contain all the available tags (or branches) of the corresponding packages; a special directory with the same name of the project with all uppercase letters is also included in the tags directory, dedicated to the tags of the project configuration directory (cmt), also called project tags within LHCb. In some cases, the project tags directories contain also copies of the packages versions. Fig. 1 shows an example of the described structure of the repository.

For new packages that do not belong yet to any project and for those that are considered obsolete, dedicated fake projects are foreseen (with the conventional names of packages and

![Image: Diagram of the LHCb repository structure]

**Figure 1.** Schema of the structure of the LHCb Subversion repository.
At the first level, the repository is divided in projects (ProjectA and ProjectB) and inside them we find the three conventional directories trunk, tags and branches. The tags directories of the packages (e.g. ProjectA/tags/PackageA1/v1r0) are copies of the corresponding trunk directories (e.g. ProjectA/trunk/PackageA1). The project tags directories (e.g. ProjectA/tags/PROJECTA/PROJECTA_v1r1) contain copies of the trunk version of the directory cmt and, optionally, copies of the tagged versions of the packages (e.g a copy of ProjectA/tags/PackageA2/v1r0 as ProjectA/tags/PROJECTA/PROJECTA_v1r1/PackageA2). The structure below the branches directory is similar to that of tags. The names of the package and project tags follow the LHCb conventions.
In order to help the tool used for the check out (see section 2.2), two special SVN properties are added to the repository top level directory: *projects* and *packages*, containing respectively the list of projects and of packages (with the project they belong to) that are hosted by the repository.

2.2. **The tools**

Once the migration is completed, users have to adapt to a new way of working.

LHCb developers were using a custom tool to check out the source code of the packages they were interested in. Part of the preparation of the migration consisted in the adaptation of this tool to understand both the old CVS repository and the new SVN one. It has proven very effective to allow the coexistence of the two repositories, because it allowed us to perform an incremental migration.

Another important tool that has been developed specifically to simplify the migration is a script used to tag the packages. It provides an abstraction over the actual repository being used and encapsulates the technical details of the tags in Subversion (essentially the paths to the source and the destination of the copy operation).

Together with the command line tools just described, the pre-commit checks of the CVS and Subversion repositories have been adapted for the migration. In the CVS repository the pre-commit check has been extended to report a meaningful message when somebody tries to operate on a package that has been already migrated. For Subversion, a pre-commit check has been to prevent unwanted commits in the tags directory, which is the most common error when porting habits from CVS to Subversion.

3. **Migration**

3.1. **Planning**

Thanks to the script that allows to transparently check out packages from CVS or from Subversion, we have been able to perform an incremental migration, i.e. migrate small groups of packages at a time.

The migration schedule of the packages was agreed with their maintainers to avoid that it happened during busy periods (e.g. just before a release).

3.2. **Procedure**

The actual migration of the files was performed using the tool *cvs2svn*[5], a program that converts the history contained in a CVS repository to Subversion commits. *cvs2svn* uses a configuration file which is written in the Python programming language, so it is very easy to write as a template where only the minimal information is inserted by hand (name of the packages/projects to migrate) and all the details (mapping from CVS path to Subversion path, paths for the tags, conversion options) are computed dynamically.

To migrate a set of packages, the librarian prepares the specific *cvs2svn* configuration from the template and tests it on a temporary local repository. After the test he announces the migration of the packages and flags them as *migrated* so that commits in the CVS repository are blocked. Then he follows the required steps to migrate the files from one repository to the other, disables the check out from CVS and enables the check out from Subversion. To finish the procedure, he announces the completion of the migration of the packages.

4. **Summary**

Subversion is an extremely versatile tool, but with its design choices makes it difficult to understand for CVS users. Because of that, it is important to encapsulate the definition of
the structure and the conventions in custom tools. This encapsulation has the added benefit of making it possible to write the tools so that they can be used with both the old and the new repository in a way transparent to the users, thus allowing an incremental migration (extremely important in case of very large repositories like the LHCb one).

The migration of the LHCb software repositories, involving the complete history of 32 projects for a total of 1082 packages and about 100000 commits, is now complete. In the first phase, starting in December 2009, we migrated the packages only on explicit request from project managers. In the second phase, that took place across July and August 2010, we migrated all the projects, except for a couple of special cases for which the migration had to be delayed until December 2010.

The CVS repository has been archived and closed.

References
[1] CVS Home Page URL http://ximbiot.com/cvs/
[2] Apache Subversion URL http://subversion.apache.org/
[3] Arnault C 2000 CMT: A software configuration management tool International Conference on Computing in High-Energy Physics and Nuclear Physics (CHEP 2000), Padova, Italy, 7-11 Feb 2000
[4] CMT Home Page URL http://www.cmtsite.org/
[5] cvs2svn Project Home URL http://cvs2svn.tigris.org/