Table of Contents

Gaudi SVN Repository **1**
- Introduction 1
- SVN Repository Organization 1
  - Rationale 1
- Version 2.0 1
  - Motivation 1
  - Structure 1
  - Meta-data (and getpack) 3
- Version 1.0 3
  - Proposal 3
- svn Quick Reference for cvs Users (version 2.0 of the repository) 5
  - Checkout 5
  - Commit 6
  - Update 6
    - Simple Update 7
    - Update with Tags 7
    - Check for updates in the repository 7
  - Comparison (diff) 8
  - Tagging and Branching 8
- References 8
  - Help desk reported problems 9

Gaudi SVN Repository

Introduction

SVN Repository Organization

Rationale

In a subversion repository, the organization of the paths is of fundamental importance for the everyday work and usability. We want to follow the standard (but not enforced) basic hierarchy of a project hosted on SVN, which consists of the 3 directories:

- `trunk`: the main trunk of development
- `tags`: fixed versions of the project identified by symbolic names
- `branches`: branches of evolution of the project parallel to the trunk

Out development model requires the necessity to have package level tags and branches. At the same time, we need something that can be used as a tag on the whole project. Moreover, LHCb unique CVS repository is hosting many CMT projects and a global tag is pointless, but we can profit from the possibility of check out a complete tagged project in one command.

Version 2.0

After having gained more experience with the usage model of the Gaudi SVN repository, I think we need to restructure it.

Motivation

Even though the previous structure is valid, it makes working with whole projects more problematic that what it could be. Mainly because of the usage of `svn:externals`, which is more powerful than the CVS aliases, but it is not flexible enough for our use case (URL pointing to the repository may change).

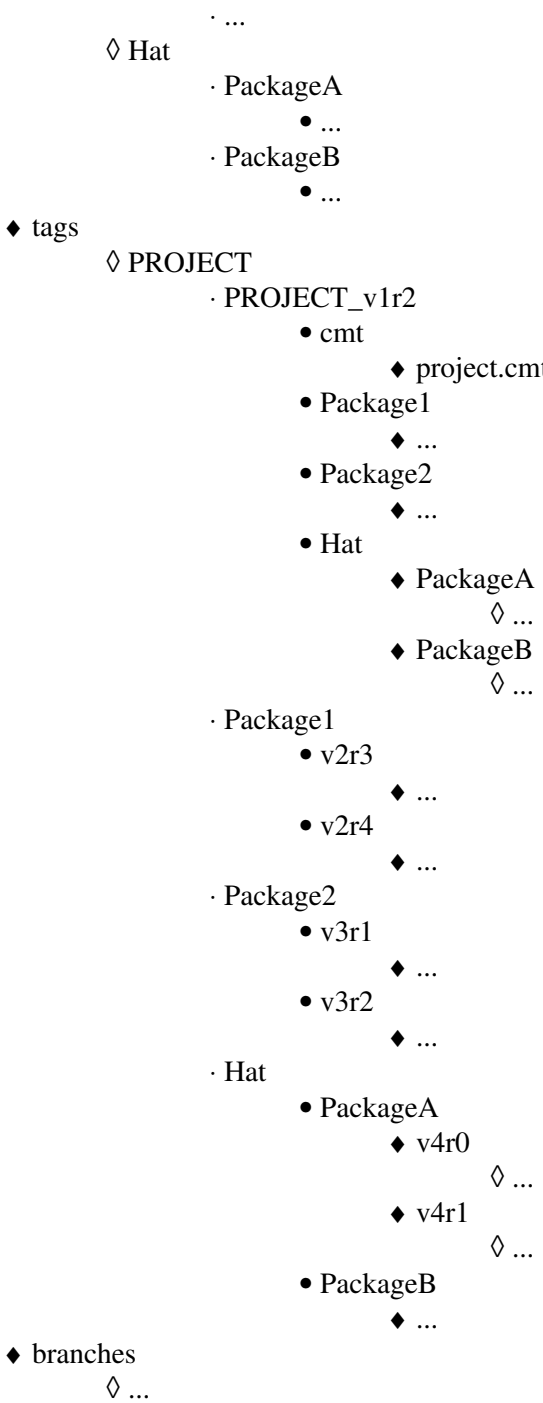
For Gaudi the package tags and the project tags have more or less the same importance (LHCb uses local tags while Atlas and other experiments use only the project tags), and for Dirac (which is being migrated to SVN too) the local tags were never used. This, essentially, means that the projects tags are more important than what was thought when devising the first structure of the repository.

The current structure makes also problematic to work with RADs like eclipse.

Structure

A structure that allows good integration between project-level and package-level tags can be achieved with something like:

- Project
  - ♦ trunk
    - ◊ cmt
      - · project.cmt
    - ◊ Package1
      - · ...
    - ◊ Package2



One directory per project (all capital letters, by convention) is visible at the top level of the repository. Inside that directory the three standard directories: `trunk`, `tags` and `branches`. The `trunk` directory contains the structure of the project as it should be checked out: a `cmt` directory and one directory per package (with or without a "hat" level).

The `tags` and `branches` directories sport the same structure. A directory with the same name of the project will contain the project-level tags/branches, while a directory per package will contain the package-level tags/branches. User tags (or branches) could be stored in further subdirectories in the normal package tags (or branches) directory, for example `PROJECT/tags/Package/username/tag`.

Not all the foreseen directories are needed for a working repository. A repository can be fully functional without project-level tags or without package-level tags.

In case the grouping of the packages on a per project basis (like in the LHCb repository) is not wanted, a fake project can be used to host all (or part) of the packages (for example called `packages`). Of course, in this case it is possible to check out a project only using some dedicated tool, like the "getpack" utility used in LHCb.

The special project `packages` can also be used to host the packages the do not belong (yet/anymore) to a project.

Note that to simplify the maintenance of the repository, when a package needs to be moved from a project to another (or to the special project `packages`), also the `tags` and `branches` directories must be moved.

Meta-data (and getpack)

To make it possible for tools to work with both version 1.0 or version 2.0 of the repository and to take into account the fact that the packages are distributed between various projects (and not grouped in a single directory) some properties of the top-level directory have to be set:

`version`
: must be set to "2.0" to declare the version of the structure of the repository

`projects`
: newline-separated list of projects in the repository, the correct case can should be used

`packages`
: newline-separated list of known packages in the repository

Empty lines or lines starting with `#` should be ignored.

A line of the `packages` property must be a space-separated list containing the name of the package (with hat), the name of the project owning it. Optionally it can contain tag specifications (a string representing a path in the repository where `%v` stands for the version id), to be used to locate special tags (useful if the package has been moved from one project to another). For example:

```
Hat/PackageA Project   OldProject/tags/OLDPROJECT/%v/Hat/PackageA
```

in this case, the tags for `Hat/PackageA` should be found with the patterns

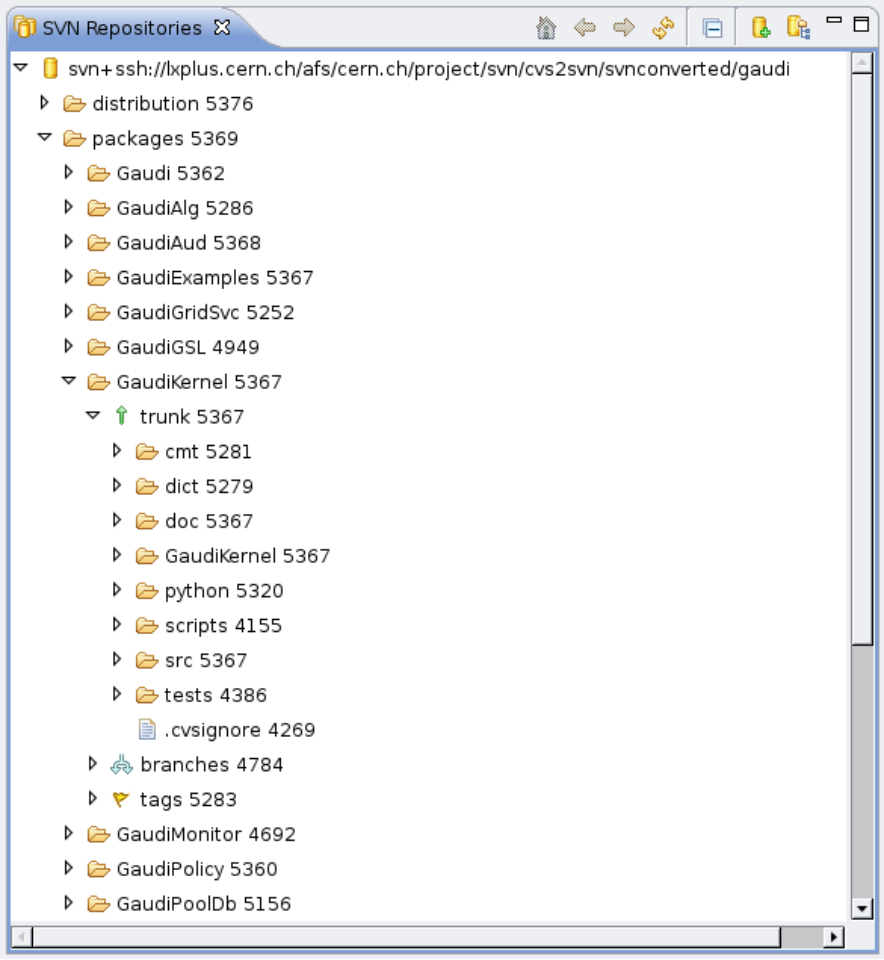
- Project/tags/PROJECT/%v/Hat/PackageA
- Project/tags/Hat/PackageA/%v
- Project/branches/PROJECT/%v/Hat/PackageA
- Project/branches/Hat/PackageA/%v
- OldProject/tags/OLDPROJECT/%v/Hat/PackageA

Version 1.0

Proposal

The repository contains the 3 main directories `packages`, `projects` and `distribution`.

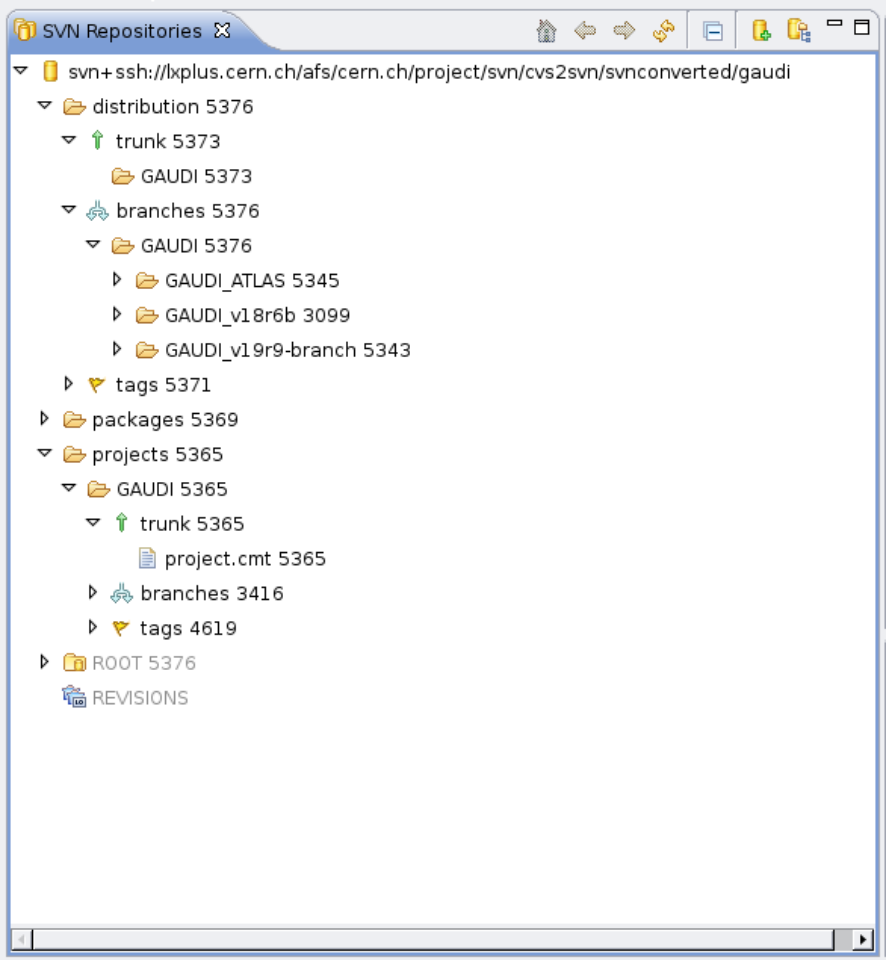
`packages` will contain one directory per package, each of them with the 3 standard directories `trunk`, `tags` and `branches`. In case of CMT "hats", we can have `packages/Hat/Package/trunk` etc. The list of known packages can be obtained scanning the content of the directory `packages`, going down one level to take into account the possibility of hats. The scanning is very inefficient, so we can add a property called `modules` to the `packages` directory containing a newline-separated list of the known modules. This property can be updated automatically with a cron job or on demand.



The `projects` directory will contain one directory per hosted project (as container of packages), with the usual 3 standard directories. The content of the trunk for one this projects is only the content of the `cmt` directory that lives at the project level:

- projects/GAUDI/trunk/project.cmt

The `distribution` directory contains the 3 standard directories. The tags and branches directory will have one subdirectory per hosted project, each of them containing one directory per "global tag". Those global tag directories will be empty and the internal structure will be described with svn externals definitions. External definitions are metadata of a directory that instruct svn to populate the directory checking out the subdirectories from other svn locations (more or less as cvs module aliases). The `trunk` directory will contain one empty directory per project, that, with the externals definitions, will allow to check out the trunks of all the related packages in one go.
Note: due to the non-locality of the external definitions (svn allows to check out from different repositories), we cannot use the same distribution sub directory for the public anonymous access and the read/write access, so `distribution/trunk/GAUDI` should be used by developers, while anonymous users will have to use `distribution/tags/GAUDI/GAUDI_HEAD` which will checkout the same files as `trunk`, but using the read-only http protocol.



svn Quick Reference for cvs Users (version 2.0 of the repository)

Environment variables for the repositories:

- GAUDICVS=:kserver:isscvs.cern.ch/local/reps/Gaudi
- GAUDISVN
  anonymous read-only access
  GAUDISVN=http://svnweb.cern.ch/guest/gaudi
  read-write access
  GAUDISVN=svn+ssh://svn.cern.ch/reps/gaudi

Checkout

- Checkout head version of a package
  - cvs

```
cvs -d $GAUDICVS checkout GaudiKernel
```

  - svn

```
svn checkout $GAUDISVN/Gaudi/trunk/GaudiKernel
```

- Checkout tagged version of a package
  - cvs

```
cvs -d $GAUDICVS checkout -r v20r0 GaudiKernel
```

- ♦ svn

```
svn checkout $GAUDISVN/Gaudi/tags/GaudiKernel/v20r0 GaudiKernel
```

- Checkout a complete project head version
  - ♦ cvs (Gaudi only)

```
mkdir GAUDI
cd GAUDI
cvs -d $GAUDICVS checkout all
```

  - ♦ svn

```
svn checkout $GAUDISVN/Gaudi/trunk GAUDI
```

- Checkout tagged version of a complete project
  - ♦ cvs (Gaudi only)

```
mkdir -p GAUDI/GAUDI_v20r0
cd GAUDI/GAUDI_v20r0
cvs -d $GAUDICVS checkout -r GAUDI_v20r0 all
```

  - ♦ svn

```
mkdir GAUDI
cd GAUDI
svn checkout $GAUDISVN/Gaudi/tags/GAUDI/GAUDI_v20r0
```

Commit

The commit is identical in CVS and Subversion

- Commit the changes in the current directory (and subdirectories)
  - ♦ cvs

```
cvs commit
```

  - ♦ svn

```
svn commit
```

- Commit the changes of a single file
  - ♦ cvs

```
cvs commit "myfile"
```

  - ♦ svn

```
svn commit "myfile"
```

Update

I want to distinguish between two use cases:

- simple update: synchronize the local copy with the latest (or a specific) revision of the repository

- update with tags: move from a tagged local version to the head one or to a branch

Simple Update

Nothing really special:

- cvs

```
cvs update
```

- svn

```
svn update
```

The way to select a revision or a moment in time a slightly different, but easy to sort out using the help.

Update with Tags

This is the complex part, since tags do not exists (as such) in Subversion. With SVN, tags are represented by copies of the `trunk` directory, so moving the working copy from a tag to another (or trunk or branch) means changing the path of the origin of the working copy.

- Move the working copy from the current tag to the trunk/HEAD (assuming you are in the top directory of the package)
  - cvs

```
cvs update -APd
```

  - svn

```
svn switch $GAUDISVN/Project/trunk/MyPackage
```

- Move the working copy from the current tag to the another tag (assuming you are in the top directory of the package)
  - cvs

```
cvs update -Pd -r other_tag
```

  - svn

```
svn switch $GAUDISVN/Project/tags/MyPackage/other_tag
```

Check for updates in the repository

To just check if there are changes in the repository that you didn't get yet, you can do a "dry-run" update with cvs. That does not exist in svn, you have to ask the status of the repository.

- cvs

```
cvs -n update
```

- svn

```
svn status -u
```

or

```
svn status --show-updates
```

Comparison (diff)

- Comparison between the base of the working copy and the working copy
  - cvs

    ```
    cvs diff -u
    ```

  - svn

    ```
    svn diff
    ```

- Comparison between two tags (or branch, or trunk)
  - cvs

    ```
    cvs diff -u -r tag1 -r tag2
    ```

  - svn

    ```
    svn diff $GAUDISVN/Project/tags/MyPackage/tag1 $GAUDISVN/PROJECT/tags/MyPackage/tag2
    ```

Tagging and Branching

- Tag the working copy (assuming you are in the top directory of the package)
  - cvs

    ```
    cvs tag the_tag
    ```

  - svn

    ```
    svn cp .  $GAUDISVN/Project/tags/MyPackage/the_tag
    ```

- Tag the head version in the repository
  - cvs

    ```
    cvs rtag the_tag MyPackage
    ```

  - svn

    ```
    svn cp $GAUDISVN/Project/trunk/MyPackage $GAUDISVN/PROJECT/tags/MyPackage/the_tag
    ```

Branching is done in CVS addind the option '-b' to the command 'tag', while in Subversion it is enough to replace `tags` with `branches`.

References

- CVS - Concurrent Versions System
- Subversion
- Subversion book (online)
- cvs2svn (tool to migrate a cvs repository to subversion)
- CERN Central SVN Service

- CERN Central SVN Service - HOW TO
- Web interfaces to Gaudi repository:
  - Trac
  - WebSVN
- CHEP2010 Paper: Migration of the Gaudi and LHCb software repositories from CVS to Subversion

Help desk reported problems

These are just to remember them.

- CT564755: pre/post-commit hooks problems ✅
- CT565111: problems with guest access to gaudi repository ✅
- CT565448: svn commit extremely slow ✅

-- MarcoClemencic - 22-Oct-2009

This topic: Gaudi > GaudiSVNRepository
Topic revision: r18 - 2012-01-06 - MarcoCattaneo