ORNL/TM-2012/41409

White Paper on data Repository Reorganization Proposal for the xLPR Project

September 2012

Prepared by
Hilda B. Klasky
Paul T. Williams
B. Richard Bass

ORNL/TM-2013/41409

Computational Sciences and Engineering Division

White Paper on Data Repository Reorganization Proposal for the xLPR Project

Author(s)

Hilda B. Klasky
Paul T. Williams
B. Richard Bass

Date Published: September 2012

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

Table of Contents

1 Executive Summary

As the xLPR project moves along, it is important to properly manage the knowledge generated by the different groups. We focus specifically on the knowledge and communications written in files, including general documents, source code and executable files. Data generated through the project are different in nature and, for this reason, need to be treated differently. To that end, ORNL put in place a series of tools that facilitate proper storage and management of project data, document and code changes, group collaboration, knowledge transfer, transparency, accountability and auditability. This paper describes the approaches/tools that we recommend for moving the project forward on knowledge management.

2 INTRODUCTION

ORNL has been given the task of managing data for the xLPR project. The data generated by the project includes general documents and code. We have put in place a series of repositories that help with the management of the data generated through the project. ORNL is hosting three main repositories for xLPR data:

1) a Subversion repository for data that requires change tracking (located at https://xlpr.ornl.gov/svn).

2) a wiki system for knowledge management (located at https://xlpr.ornl.gov/wiki), and

3) a shared drive (located at https://xlpr.ornl.gov/share) to facilitate transfer of huge files that do not require change management and cannot conveniently be shared through other means.

In the following sections we describe the latter repositories, the reason why we recommend them, a proposal for the directory structure and what to store on each repository.

3 xLPR Subversion repository structure.

The xLPR Subversion repository uses the standard repository layout suggested in the Subversion documentation at http://subversion.apache.org/docs/. The basic elements of the xLPR Subversion repository are three directories: 1) trunk, 2) branches, and 3) tags. The latter directories in Subversion can be checked out separately. Figure 1 is a visual representation of a Subversion repository layout. The *green* items represent the evolving flow of the *trunk* directory activity. The *yellow* items represent the evolving flow of the *branches* directory activity. The *blue* items represent the *tags* directory activity. Trunk, branches and tags directories are explained below. Please bear in mind that each node in Fig. 1 represents a copy of the whole Subversion repository.

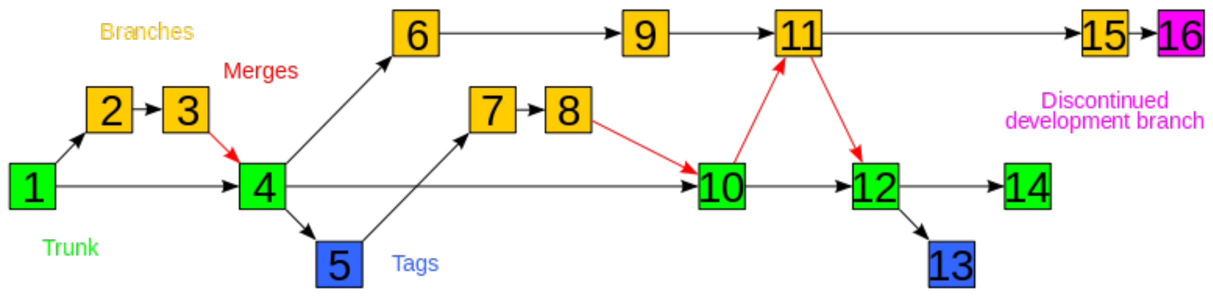

Fig. 1 Visual representation of a Subversion repository layout

1. The ‘trunk’ directory holds the “main line” of development (See green items in Fig. 1). The trunk contains the most current development code/document at all times. This is where users work up to their next major release of code. The trunk should only be used to develop code that will be the next major release.

2. The ‘branches’ directory contains branch copies of the trunk directory (See yellow items in Fig 1). With the branches directory, users can create paths for their code/documents to progress to more specific goals, like an upcoming release. The branches directory contains copies of the trunk at various stages of development.

3. The 'tags’ directory contains tag copies (See blue items in Fig. 1). Tags are, like branches, copies of your code. Tags, however, **are not to be used for active development**. They mark (tag) a certain state of your code. It is a snapshot of your deliverables at a certain point in time.

Additional details are provided in Section 9 (Annexes).

4 What to store in Subversion?

Data that needs to be baselined, i.e. ‘tagged’, should be stored in Subversion. Subversion is exactly the right tool for:

- archiving old versions of files and directories, possibly resurrecting them, or examining logs of how they've changed over time
- collaborating with people on documents (usually over a network) and keeping track of who made which changes
- tracking changes on source code

This is why Subversion is so often used in software development environments—working on a development team is an inherently social activity, and Subversion makes it easy to collaborate with other programmers. Of course, there's a cost to using Subversion as well: administrative overhead. ORNL will manage a data repository to store the information and all its history, and be diligent about backing it up. When working with the data on a daily basis, users won't be able to copy, move, rename, or delete files the way you usually do. Instead, users have to do all of those things through Subversion.

Now, we should be aware that using a Subversion repository adds extra workflow to the project. And, users must make sure they are not using Subversion to solve a problem that other tools solve better. For example, because Subversion replicates data to all the collaborators involved, a common misuse is to treat it as a ***generic distribution system***. People will sometimes use Subversion to distribute huge collections of pdf files, photos, digital music, or software packages. The problem is that this sort of data usually isn't changing at all. The collection itself grows over time, but the individual files within the collection aren't being changed. In this case, using Subversion is “overkill”, or it is like swatting a fly with a Buick.

Using Subversion to store a folder with technical reports and documents that do not change over time will only slow the process of accessing the xLPR repository and will get us into difficulties at ORNL for not using Subversion as a tracking tool but as a generic distribution system. We are not concerned so much about storage size, but about network bandwidth issues. **Each user action - performing a check out, creating tags or branches, etc. - will make complete copies of the whole repository, and the accumulation of these copies over time could overwhelm the network!**

The xLPR Subversion repository should be used to store project documents and code that need version tracking, i.e. the Configuration Items (CIs) described in the Software Configuration Management Document. For the xLPR repository,

- the ‘trunk’ root directory contains subdirectories for the different xLPR ‘tasks groups’.
- Within each ‘task group’ subdirectory, there are two main folders:
  - the “docs” folder which contains documents, and
  - the “src” folder which contains source code.

So, a sample task group should contain the following:

```
task_group_name/docs/deliverables

                /src/
                    /conf
                    /libs
                    /…
```

5 Where to store the rest of the xLPR documents?

ORNL has set up an xLPR wiki at https://xlpr.ornl.gov/wiki. A major objective here is to prevent a network bottleneck that could result from copying large amounts of files that do not change over time when tagging and branching the repository. In the wiki, users will find relevant reports/documents, such as MRPs, NUREGs, PVP papers and other publications. Also, the wiki is useful for storing documents that do not change over time and that do not require software configuration management; examples are pdf files, images, meeting minutes, and PowerPoint presentations.

xLPR users should not fear the learning curve associated with use of the wiki. That effort is comparable to learning Subversion through the TortoiseSVN client. Wikis are effective tools for content management and are replacing shared drives for reasons that include:

- Searching documents and file contents is easy
- Attachments are versioned
- Changes to web pages can be tracked
- Web pages can be commented and users can see the associated meta data

- Easy creation of image thumb-nailing and galleries
- Availability of tools to facilitate group collaboration

The xLPR wiki is a private storage for internal documentation, communications, and dissemination of information across institutional and national boundaries.

Moreover, other important projects and research groups are already making use of wikis to share project documents. Examples of those include

In the public arena:
- http://moinmo.in/
- http://twiki.org/
- http://code.google.com/

In the private arena:
- MS SharePoint
- Atlassian Confluence
- IBM Connections

Because of all the advantages that a wiki provides to the team, we suggest using it to store meeting minutes, progress reports, presentations, reference documents and white papers. The following tree presents the basic structure of the xLPR wiki in task group space:

```
task_group_name/meetings
               /monthly_reports
               /presentations
               /reference_docs
               /white_papers
```

xLPR Groups should at least use the meeting_minutes, monthly_reports and presentations folders. xLPR Groups can create other folders as needed, for example: QA Templates is a folder needed for xLPR QA template documents.

Why do we recommend a wiki system over a shared drive for xLPR documentation? Because wiki systems are more capable to perform searches, group collaboration, knowledge transfer, transparency, accountability and auditability. This is the reason why wikis are now replacing shared drives. The disadvantages of using shared drives to store project data are:

•**Structure** – Because they're so simple, there's no structure. Vast forests of folders spring up and people aren't generally sure where to put things anymore. They find their own little corners of the drive, and just put all files there.

•**Gardening** – People are afraid to delete anything because they didn't put it there. Someone else stored the file, so I'm not going to delete it – they might want it.

•**Search** –Quite simply, for 90% of share drives (probably 99%), there is no search. Google boxes and other tools can solve this problem, but most people don't have them on the "server in the corner".

•**Versioning** – How are files versioned in a typical shared drive? Renaming! You know what I mean. Which file is newer, i.e., specification-17Nov12.doc, specification-v2.doc, specification-mikes-edit.doc, specification-draft.doc? No one knows. In the end you probably look for the file modification date, but that can be dangerous.

•**History** – Who changed what and when? There are no names with file changes and no comments ("I've edited this and it's good to go."), so it can take a long time to work out exactly what has changed.

However, ORNL does provide a shared data area to share huge files that do not need to go through the software configuration management process. For browsing/downloads, users can map a network drive to scfm.ornl.gov/share or xlpr.ornl.gov/share. Both addresses currently point to the same area.

To prevent the gardening issue mentioned above, the shared drive clean-up maintenance is as follows:

Every night, a script removes all files that have not been modified in the last 30 days, and removes all directories that have not been modified in the last 30 days if they are either 1) empty or 2) do not contain files that have been modified within the last 30 days.

6 Closure

ORNL's goal is that the knowledge management tools presented in this white paper will provide substantial added value to the xLPR consortium. Specific points emphasized here are given as follows:

- Data generated through the project are different in nature and, for this reason, need to be treated differently.

- The recommended tools will facilitate management of

  - project's documentation,
  - communications,
  - document and code changes,
  - group collaboration,
  - knowledge transfer,
  - transparency,
  - accountability,
  - auditability, and
  - dissemination of information across institutional and regional boundaries.

To seek help with your questions and problems regarding the content of this white paper, please contact Hilda Klasky at klaskyhb@ornl.gov.

7 Access to tools

- Subversion: https://xlpr.ornl.gov/svn
- Wiki: https://xlpr.ornl.gov/wiki

- Shared Drive: xlpr.ornl.gov/share

8 Links for more information

- http://svnbook.red-bean.com/en/1.7/svn-book.html
- http://en.wikipedia.org/wiki/Apache_Subversion
- http://ariejan.net/2006/11/24/svn-how-to-structure-your-repository
- http://ariejan.net/2006/11/21/svn-how-to-release-software-properly/
- http://rebelutionary.blogs.atlassian.com/2007/02/enterprise_wikis_replace_share
  d_drives_c.html

9 Annexes

9.1 Types of Subversion Branches:

The ‘branches’ directory contains branch copies of the trunk directory (See yellow items in Fig 1). With the branches directory, users can create paths for their code/documents to progress to more specific goals, like an upcoming release. The branches directory contains copies of the trunk at various stages of development. There are different types of branches. Below we present some of the most common ones.

a) **Release Branches**
When the trunk reaches the stage that it's ready to be released (or when users want to freeze the addition of new features), users create a release branch. This release branch is just a copy of the current trunk code.

The branch can be checked out separately and the user can start branding and versioning the project. The user can also employ the release branch to fix bugs that pop up during (beta) testing. The idea of this approach is to keep development progressing in the trunk without having to deal with release-specific issues. So it's perfectly fine to add new features to your trunk while you (or others) prepare the release.

b) **Bug fix branches**
Branches may also be used to address the more serious bugs found in the trunk or in a release branch. The bugs are of such magnitude that the user can't fix them in a single commit. So, to focus on the problem of fixing this bug, the user should create a new branch for this purpose. This allows development in the trunk or in the release branch to continue, without disturbing them with new bugs or tests that break the current code.

Bug fix branches are named after the ID they are assigned in xLPR’s issue tracking tool, JIRA. Typically, this ID is a number, for example: xlpr-123. Of course, the user can access bug-fix branches like any other.

c) **Experimental branches**
Experimental branches are used to try new technologies, solutions or approaches without compromising the entire project. Something that happens often is the introduction of new technologies. This is fine, of course, but you don't want to bet your entire project on the outcome.

For example, imagine that you want to change from PHP 4 to PHP 5 (PHP is a programming language) for your software tool. How long would it take you to convert your entire project? Do you want your entire code base (trunk) to be useless until you have converted all of your code? Probably not!

In this experiment, if implementing PHP 5 is a bridge too far for your application, then the latter effort should be given its own branch. You can hack your way to PHP 5 conversion on that branch and, if you fail, you still have your current PHP 4 code in the original branch.

Experimental branches may be abandoned when the experiment fails. If they succeed, you can easily merge that branch with the trunk and deliver your big new technology. These branches are usually named after the relevant experiment. I always prefix them with 'TRY-, for example:

https://svn.example.com/svnroot/project/branches/TRY-new-technology

9.2 Types of Subversion Tags

The 'tags' directory contains tag copies (See blue items in Fig. 1). Tags are, like branches, copies of your code. Tags, however, **are not to be used for active development**. They mark (tag) a certain state of your code. It is a snapshot of your deliverables at a certain point in time. There are also different types of tags:

a) Release tags

Release tags mark the release (and state) of your code at that release point. Release tags are always copies of the corresponding release branch. Release tags are prefixed with 'REL-' followed by a version number.
Users can access these tags easily:
https://svn.example.com/svnroot/project/tags/REL-1.0.0

b) Bug fix PRE and POST tags

When you have created a bug fix branch, you want to mark (tag) the status of your code before and after the bug fix. This allows you to easily refer to the changes you made when you want to merge them back to your trunk or release branch.
The start-tag is called 'PRE' and the end-tag called 'POST'. Of course, you should add the bug ID number here to show what bug you are tagging.
You probably will not check out bug fix tags, but you want to reference them when merging bug fixes with your other code:

https://svn.example.com/svnroot/project/tags/PRE-3391
https://svn.example.com/svnroot/project/tags/POST-3391