International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 06 | June-2018 www.irjet.net p-ISSN: 2395-0072

Evolution of Version Control Systems and a study on TortoiseSVN

Pratik P Bhoir[1], Harshali Patil[2]

[1]Student, Mumbai Educational Trust, Mumbai, Maharashtra, India
[2]Professor, Mumbai Educational Trust, Mumbai, Maharashtra, India

---

Abstract - Almost each software engineer who is working on a closed or open source project has a critical problem – managing their work. An open source project is contributed by large number of software developers. In such cases, it is quite difficult to keep track of the changes made to the source code because there can be a malicious developer whose major aim is to damage the project. So, these kinds of harm should be identified as quick as possible and there should be a way to revert back to older working solution in case of any failure. Each working version of the file should be recorded for the sake of the user to recoup effortlessly and the user can switch to any working version anytime.

Key Words: CVCS, DVCS, GIT, TortoiseSVN.

1. INTRODUCTION

This document is template. We ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download the template and replace(copy-paste) the content with your own material. Number the reference items consecutively in square brackets (e.g. [1]). However, the authors name can be used along with the reference number in the running text. The order of reference in the running text should match with the list of references at the end of the paper.

2. History of Version control systems

This document is template. We ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download the template and replace(copy-paste) the content with your own material. Number the reference items consecutively in square brackets (e.g. [1]). However, the authors name can be used along with the reference number in the running text. The order of reference in the running text should match with the list of references at the end of the paper.

2.1 The Emergence

To overcome this problem, Source Code Control System (SCCS) came into play. The SCCS was designed to track changes in the source code and other text files during the software development. It was developed at Bells Lab in 1972 by Marc Rochkind and it was implemented on IBM System/370 computer running OS/360. Soon later Revision Control System (RCS) replaced SCCS. RCS was first released in year 1982 at Purdue University by Walter F. Tichy. It operated only on single files., hence it did not support atomic commits affecting multiple files. The RCS structure was very simple to understand and easy to work with. But security was an issue and only one user could work on a file at a time. [6].

2.2 The Classic Phase

In 1990, VCS moved into new phase. The Concurrent Version System (CVS) was developed by Dick Grune as a series of shell scripts. CVS uses a Client-Server architecture. The current version of the project(s) and its history are stored in the server. The clients are connected to the server to "check out" the complete copy of the project, work on the copy and later on "check in " the updated copy of the project. The CVS is a type of Centralized Version Control System (CVCS) which allowed many developers to work on the same file at the same time. It was the first which supported Merging and Branching but it was didn't worked efficiently. [6]

2.3 The Post-Classic Phase

Post classic phase of the VCS, came the concepts of Subversion (SVN). It is the open source Centralized version control system under the Apache Licence . It was created by CollabNet Inc. in 2000 and is still widely used globally.

It maintains the versioning for files, directories, the file metadata and the renames. The users can copy and/ or move entire directory-hierarchy very quickly and easily, while retaining full revision history. In 2010, it became one of the top-level Apache project. The latest stable version of Subversion (1.9.0) was released in 2017. [6]

2.4 The Modern Phase

There came a renaissance in the year 2009-2010 with the introduction of GIT and its like. GIT is a Distributed Version Control System (DVCS). Linus Torvalds developed GIT in year 2005 with an aim to increase speed, data integrity and support for distributed and, non-linear workflow. It is a free open source software under the terms of GNU General Public License version 2.

GIT provides a support for non-linear development of the software. It supports quick branching and merging and also includes tools for navigating and visualizing a non-linear development history.

GIT is compatible with existing systems and protocol. The repositories of GIT can be published via Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), or a Git protocol over either a plain socket, or Secure Shell (ssh).

GIT is efficient of handling large projects. It is very fast and more scalable than other few versions of VCS. It can fetch version history from the locally stored repository hundred times faster than fetching it from the remote server. GIT can perform automatic garbage collection when there are enough of loose object creation in the repository.

3. Types of Version control systems

Version Control systems provide a way to manage a large number of developers working together and keep a track of who does what. Over the ages, there is an evolution in the Version Control Systems. The Version Control System can be categorized into three types: Version Control systems provide a way to manage a large number of developers working together and keep a track of who does what. Over the ages, there is an evolution in the Version Control Systems. The Version Control System can be categorized into three types:

3.1 Local Version Control System

Majority of people choose version-control method is copy files into another directory (better if the directory is time-stamped). This is the commonly used approach as it is very simple, but at the same time, it is also incredibly error prone. There are higher chances that the user may forget the current working directory and he/she may accidentally write to the wrong file or copy over files which you don't mean to.

To solve this issue, programmers have developed Local VCS's that have a simple database that keeps changing to files under revision control system

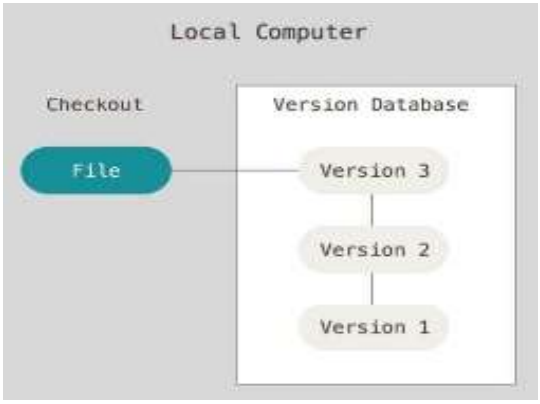

Fig -1: Local Version Control System

RCS is one of the popular Local VCS, which is still distributed with many computer systems today. The famous operating system Mac OS X still includes the RCS command when installed with the Development tools. RCS has a straightforward interaction model. For each working file or document, you instate its RCS file once, at a point enter a cycle of checkout, change and checkin tasks. En route, you can change a portion of the RCS record's metadata, also. The greater part of this is done through RCS commands; you require not alter the RCS record straightforwardly (and in actuality, you ought to most likely abstain from doing as such or RCS end up confused). This model is fairly comparable to utilizing a library (of books). With a library, you agree to accept a library card (instate), at that point enter a cycle of taking a book home (checkout), appreciating it (NB: without alteration, one expectation), and returning it to the library (checkin).

3.2 Centralized Version Control System

People encountered issues when they needed to collaborate with the developers on the other systems. To overcome this problem, Centralized Version Control System were developed. These types of VCSs record the history of changes on a central server from where everyone requests for the latest version of the work and pushes in the newest changes to. Everyone sharing the server also shares everyone's work.

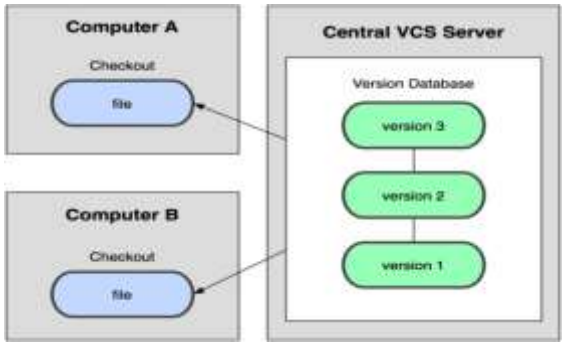

Fig -2: Centralized Version Control System

There are few famous tools available to meet this need, For example: CVS, Perforce and SVN. These types of VCSs allow more access control via folder permissions by allowing checkout of the only needed sub tree from the repository tree. These systems are more prone to be backed up regularly, providing more reliable environment of work.

Centralized Version Control System overcomes the problem of previous type of VCS of working on a single file at a time and most of the other drawbacks. However, this system also has some serious downsides. If that server goes down even for some time, then during that duration no user can merge their work at all or save versioned changes to anything they're working on. If the central database becomes corrupted, and proper backups haven't been kept, everything is lost.

3.3 Distributed Version Control System

To overcome the drawbacks of Centralized VCSs, Distributed Version Control Systems stepped in. In DVCS, the client users not only check the latest versions of the files, but also

mirrors the repository. So, in case if any servers goes down, and these systems are collaborating via this server, any of the client repositories can be copied back to the server to restore it. Every checkout is a full backup of all the data.

Many DVCS work pretty well with having many remote repositories they can deal with, so the developers can work with different groups of people in various ways working within the same project. This enables them to set up several kind of workflows that aren't possible in the centralized version control systems, such as hierarchical models.

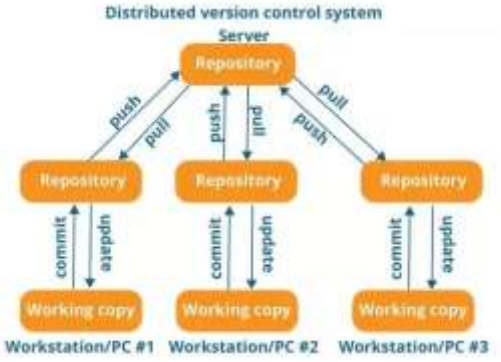

Fig -3: Distributed Version Control System

The Distributed VCS come with advantages like availability. The history is fully available to everyone at all the times. It is extremely fast because of its local nature of the majority of the operations. It doesn't require access to remote servers. Branching and merging can be done very easily in Distributed VCS.

4. ABOUT TORTOISE SVN

TortoiseSVN is an open source Windows client for the Apache Subversion version control system which is available at free of cost. The files are stored in the central repository. The repository is just like an ordinary file server, except that it records every change made to your files and directories. This gives users a freedom to recover to the older versions of the files and analyze the history of how and when your data changed and who changed it.

TortoiseSVN is very easy to use as all the commands are available directly from the windows explorer. User can view the status of their files and directories directly in the windows explorer. It also provides descriptive dialogs and is constantly improved due to user feedback. The subversion protocols that are supported are http, https, svn, svn+ssh, file, svn+XXX. TortoiseSVN also provides mechanism which allows to integrate any web-based bug tracking system.

4.1 FEATURES OF TORTOISE SVN

4.1.1 SHELL INTEGRATION

It integrates smoothly into the Windows shell (the explorer). The user does not need to switch to different application when he need functions of version control. The user is not limited to using the Windows Explorer, TortoiseSVN's context menus work in the other file manager. All the subversion commands are available from the explorer context menu. TortoiseSVN adds its own submenu in the explorer context menu. [5]

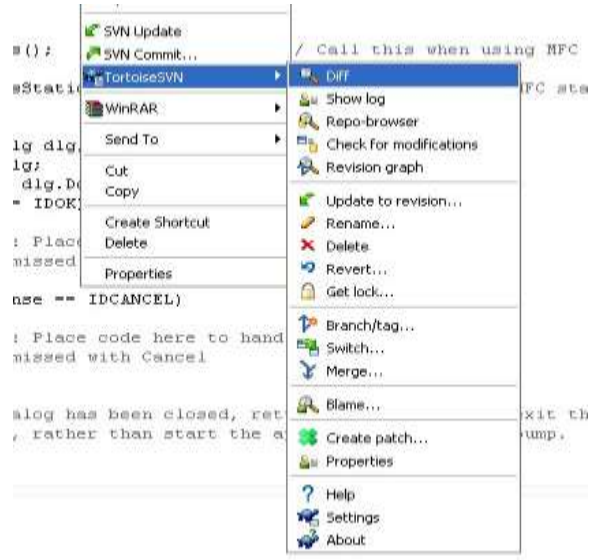

Fig -4: TortoiseSVN Context Menu

4.1.2 iCON OVERLAYS

The status of each and every versioned file is indicated by small overlay icon. The user can view the status of the working file right from it.

The different status of the file or directory can be:

Normal, readonly, added, deleted, modified, locked, conflicted, ignored, non- versioned. [5]

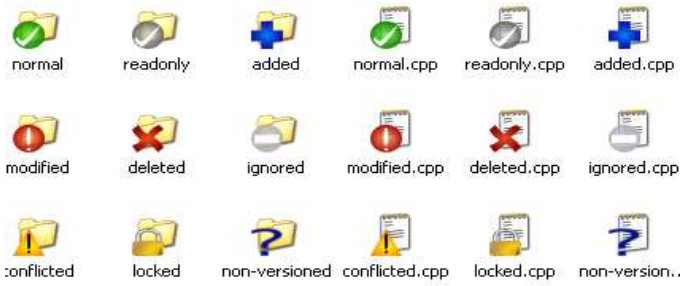

Fig -5: TortoiseSVN icon overlay

4.1.3. GRAPHICAL USER INTERFACE

TortoiseSVN helps user to view a list of changed files. The commit dialog displays all the items that will be in included in the commit, and each file/directory has checkbox so that user can choose which file/directory want to be included. [5]

4.1.4. COMPLETE COMMIT

A commit command invoked on a file goes into the repository completely, or not at all. This enables developers to build and commit changes as logical chunks. [5]

4.1.5. VERSIONED METADATA

The file differences are expressed by subversion using binary differencing algorithms, which functions identically on the both text and binary files. The both types of files are stored equally compressed in the repository, and the differences are sent in both the directions across the network. [5]

There is a set of "properties" attached to each file and directory. These properties are versioned over the time, just like the file contents. User can invent and store any arbitary key/value pairs the user wish. [5]

4.1.6. WELL ORGANIZED BRANCHING AND TAGGING

Branches and tags are created easily by subversion by simply copying the project, using a mechanism similar to a hard link. These operations require very less and constant time, and occupies very less space in the repositories.

4.2. BASIC VERSION-CONTROL CONCEPTS

4.2.1 THE REPOSITORY

TortoiseSVN is a centralized version control system. The repository is at its core, which is the central store of the data. The data is stored in the repository in form of filesystem tree which is a typical hierarchy of files and directories. Multiple clients can connect to the repository and can read or write to the files. A client makes the information available to others by writing data. The client receives information from others by reading the data. [3]

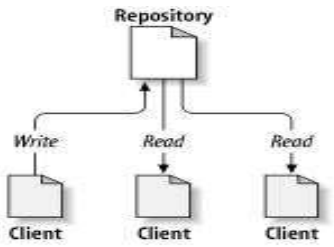

Fig 6 –: A Typical Client/Server system

The Subversion repository records each and every change ever written to it. It remembers every change to every file, and even changes to the directory tree itself, activities such as addition, deletion, and rearrangement of files and directories.

When a user reads a file from the repository, the user not only sees the latest version of the filesystem tree but also view previous states of the filesystem. The TortoiseSVN has an ability to answer historical questions, "what this file contained last Monday?", or "what changes were made to the file?", and "who the last person was to make changes in the file?" These are the kind of questions which can be easily answered by TortoiseSVN.

4.2.2. THE VERSIONING MODEL

The main aim of all VCSs is to solve a common problem: How will the system will allow the developers to share information by avoiding concurrency.

Lets imagine a scenario: Two co-workers Alice and Bob decide to edit the same file in the repository at the same time. If Bob saves his changes to the repository first, then there are chance that after a few moment later Alice could accidentally overwrite them with his own new version of file. The Bob's version of the file won't be lost permanently as the system records every change, but changes made by Bob won't be present in Alice's version of the file, as she never saw Bob's changes before updating the file. But Bob's work is still lost from the latest version of the file accidently. This is the situation every team needs to avoid while working on a project. [3]

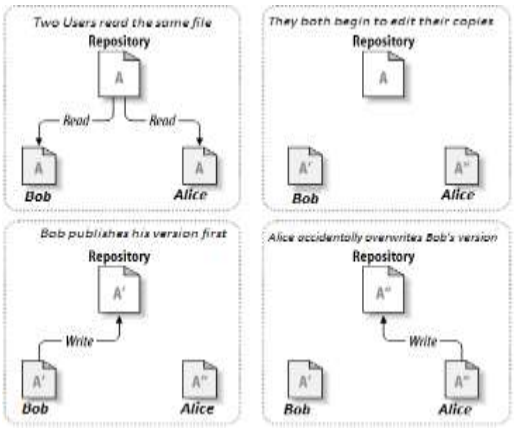

Fig 7 :- Problem to be avoided

4.2.2.1. THE LOCK-MODIFY-UNLOCK SOLUTION

Most of the Version Control Systems use the lock-modify-unlock solution for the above problem, and it is considered as the simplest solution. But in such system, the repository allows only one user to modify a file at a time. First Bob must lock the file so that he can make modifications in it. Locking the file while prevent Alice from making changes to it. All Alice can do is read the file and wait for Bob to release the lock. If she tries to lock the file, the repository will reject the request. After Bob has released the lock, Alice can view Bob's

work in the file and only then she can apply lock to the file to edit it. [3]

The issue with lock-modify-unlock solution is that it is restrictive and it often becomes a blockage for users. The various problems which arises are:

1. Administrative problem
2. Unnecessary Serialization
3. Create a false sense of security

4.2.2.2. THE COPY-MODIFY-MERGE SOLUTION

TortoiseSVN use this copy-modify-merge solution as an alternative to locking. In this method, each user reads repository and creates a working copy of the file or project which is personal. This enables users to work parallel and modify their personal copies. In the end, all the personal copies are merged into a new, final version. [3]

Consider another scenario: Alice and Bob each one of them create a working copy of the same project, which is copied from the repository. Both work concurrently, and make changes to the same file A, but within their own copies. Alice saves her changes to the repository first. When Bob tries to save his changes after Alice, the repository informs Bob that the file has been somehow changed since he last copied it. So, Bob asks his colleagues to merge any new changes from the repository into his working copy. Once the set of changes are integrated to his copy, he saves his working copy back to the repository.

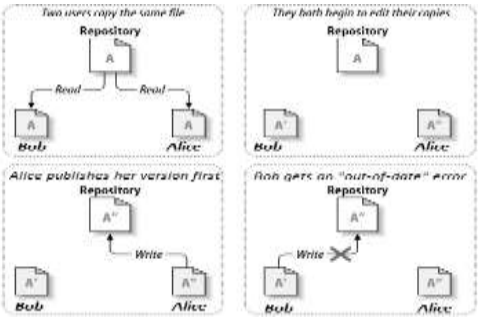

Fig 8- The COPY-MODIFY-MERGE Solution

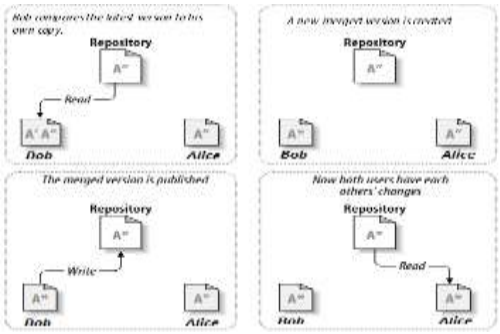

Fig 9- COPY-MODIFY-MERGE Solution (cont.)

This solution may sound a bit confusing, but practically it runs effectively smoothly. This allows users to work in parallel and never wait for one another. When multiple users work on the same files, most of their concurrent changes don't overlap at all.

TortoiseSVN by default uses this copy-modify-merge solution. There is only one situation where lock-modify-unlock method comes out better than this solution, that is when there are unmergeable files. For example, if users are working on some graphic images and both the users make some changes in that image at the same time, there is no possibility by which both the images can be merged together. Either user A or user B will lose their changes.

4.2.3. TORTOISE SVN IN ACTION

4.2.3.1. WORKING COPIES

A TortoiseSVN working copy is nothing but an ordinary directory tree on the user's local system, which contains the collection of files. User can edit these files and if the files are source code files, user can compile the program from them in the usual way. The working copy of the user is his/her own private work area. TortoiseSVN would add other people's changes, nor make user's changes available to others, until the user explicitly orders it to do so.

When the user has made changes to the files in the working copy and verified it whether it works properly, TortoiseSVN provides user with commands tp publish the work to the rest of the team which is working with him on the same project (by writing to the repository). If other team member publishes their own work, TortoiseSVN provides user with the commands to merge those modifications into the users working directory (by reading from the repository).

There are also few extra files, created and maintained by TortoiseSVN in the working copy, which helps users to carry out those commands. The working directory contains a subdirectory named .svn , also known as the administrative directory of the working copy. The administrative files helps TortoiseSVN to recognize files with unpublished changes and the "out-of-date" file with respect to others work.

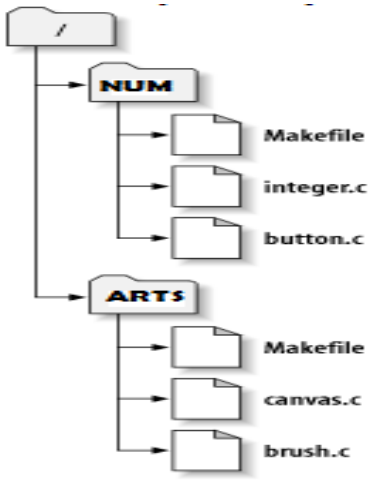

Fig 10 – Repository's filesystem

The repository's root directory contains two sub-directories: NUM and ARTS. To create a private copy, i.e working copy of the project, user must checkout some subtree of the repository. If the user makes some modifications in the file integer.c, subversion command commit can be used to publish the updated file to others. To bring the project up to date, user can use the command update. This will incorporate all the changes committed by whole team in their personal copies in the users working copy.

4.2.3.2. REPOSITORY URLs

There are different methods by which TortoiseSVN repositories can be accessed - on a local disk, or through different network protocols. A repository location, is always an URL. The below URL schema indicates the access methods:

SCHEMA	ACCESS METHODS
file://	Direct repository access on the local or network drive
http://	Access via WebDAV protocol to subversion-aware Apache server
https://	Same as http://, but with SSL encryption.
svn://	Unauthenticated TCP/IP access via custom protocol to a svnserve server.
svn+ssh://	Authenticated, encrypted TCP/IP access via custom protocol to a svnserve server.

Table -1: Access method schema

4.2.3.3. REVISIONS

Many files and directories can be changed by the commit command operation as a single atomic transaction. A user can change content, create, delete, rename and copy files and directories, and then commit all these operations together as a unit. Each commit command is treated as an atomic transaction. Every time the repository accepts a commit, a new state of the filesystem tree is created, which is called a Revision. Each revision is identified with a unique natural number, the number is greater by one than the number of previous revision. The first revision of the freshly created repository is numbered zero as it an empty root directory.

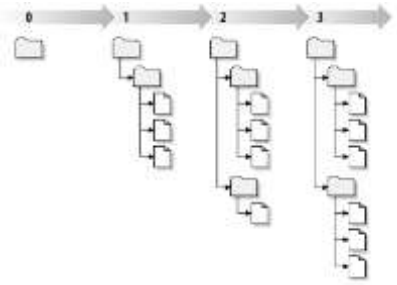

Fig 11- The revisions of the Repository

4.2.4. BASIC TORTOISE SVN OPERATIONS

4.2.4.1. CHECKOUT

To checkout a working copy from the repository user need to do:

Select the directory from the windows explorer where you want to place your working copy. Right click on that to get Tortoise context menu. Select TortoiseSVN → Checkout, which will pop up the below dialog.

Fig 12- TortoiseSVN Checkout in Windows OS.

4.2.4.2. COMMIT

Reflecting the changes, you made in the working copy to the repository and make it available to the other users who are working on the same project can be done using commit command. But make sure before committing your working copy is up-to-date. To update you can use from the context menu TortoiseSVN → Update directly.

If your working copy is up-to-date then you are ready to commit the changes. Select the file/ folder you want to

commit, then select from the context menu TortoiseSVN→Commit..

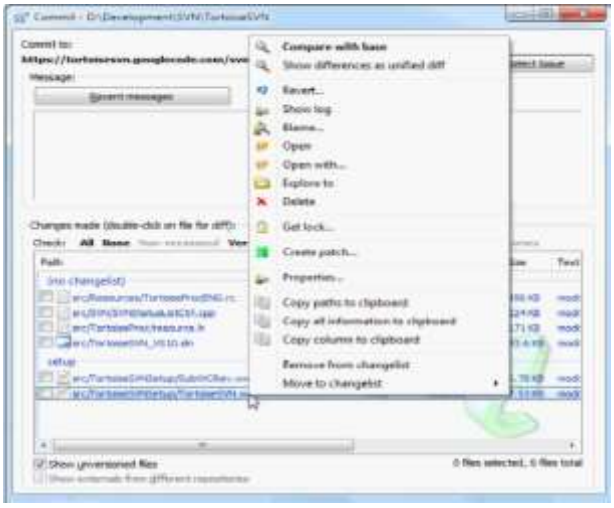

Fig 13- TortoiseSVN Commit in Windows OS.

4.2.4.3. UPDATE

To update any file / folder in the working copy, right click on the file/ folder which needs tobe updated and select TortoiseSVN →Update from the explorer menu. This will update your working copy with the changes from others.

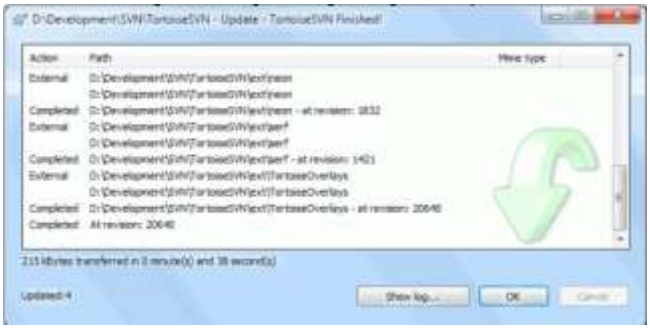

Fig 15- TortoiseSVN Update in Windows OS.

If you want to update the file/folder to a certain revision then you should use TortoiseSVN → Update to Revision..

4.2.4.4. REVISED LOG DIALOG

User should provide a log message from every changes made and committed to a file. This way user can keep a track why and when the changes were made, and he has a detailed log file to answer these questions.

The Revision Log Dialog answers all queries about the revision of the file. It displays the revision of file, the Author who committed the changes, date – on which the changes were committed, and the log message – which tells why the changes were made.

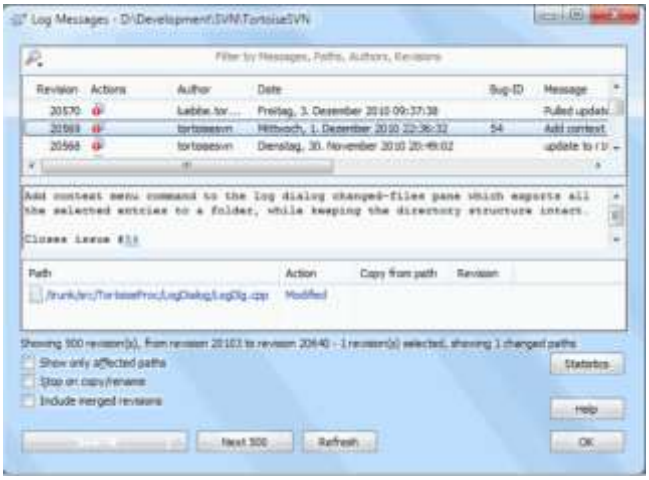

Fig 16-TortoiseSVN Revision log Dialog in Windows OS..

5. CONCLUSIONS

The development of source control system was not started with the many types of technology. They simply maintained a database with records of files and its versions. But as time progressed Version Controlling played an important role in software engineering. Thus, centralized VCSs like TortoiseSVN came into existence. TortoiseSVN made version controlling very easy and brought a change in software engineering. But soon the Centralized VCSs will be replaced by Distributed VCS to improve the performance and scalability. Though many organizations now have shifted to Distributed Version Control Systems like Git, TortoiseSVN is still in use and is considered reliable by many portals and organizations like SalesHoo.com, Affilorama Affiliate Marketting, Thomson.com, VoteHere, etc.

REFERENCES

[1] Version Control Systems for Corporations: Centralized and Distributed An explorative case study into the corporate use of version control systems" My Högblom ,Viktor Green

[2] N. B. Ruparelia, "The history of version control", ACM SIGSOFT Software Engineering, 2010, 35,

[3] "Version Control with Subversion", Ben Collins-Sussman Brian W. Fitzpatrick C. Michael Pilato

[4] B. de Alwis, J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?", In Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering, IEEE Computer Society, 2009,

[5] https://tortoisesvn.net/docs/release/TortoiseSVN

[6] "Version Control System – GIT", Sanjay Murali, Vivek Chandru