**White Paper** 

**SVN Auto Code Commit**

Authors: Naga Panguluri, Surya Gunupudi

Date: 20th Oct 2019

Contents

[Introduction 3](#_Toc23349472)

[Approach 5](#_Toc23349473)

[Auto Code Commit 5](#_Toc23349474)

[Process 5](#_Toc23349475)

[Please find the flow diagram below 6](#_Toc23349476)

[Understanding config file 7](#_Toc23349477)

[Important locations and files 8](#_Toc23349478)

[Enabling and disabling Auto Commit 8](#_Toc23349479)

[Developer's Stand point 9](#_Toc23349480)

[POC and implementation 9](#_Toc23349481)

[Pre-Requisites 9](#_Toc23349482)

[Features 10](#_Toc23349483)

[Best Practices 10](#_Toc23349484)

[Limitations 12](#_Toc23349485)

[Summary 12](#_Toc23349486)

# Introduction

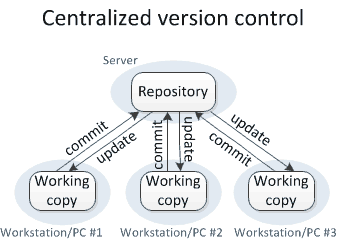
In software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle (SDLC). The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

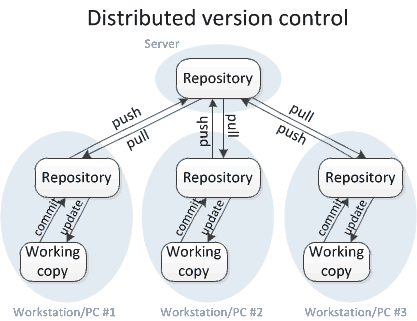
Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

In the process of development, every project maintains source code with the help of version control tools like SVN, git, perforce … etc. A little glance on version control:

A version control system is a special application that stores and manages every revision of your files and code. Many developers and organizations use version control to collaborate on source code, manage releases, and roll back to previous versions when bugs are discovered. Some of the common terminology:

* Branch 🡪 A set of files under version control may be branched or forked at a point in time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other.
* Change 🡪 A change (or diff, or delta) represents a specific modification to a document under version control. The granularity of the modification considered a change varies between version control systems.
* Change list 🡪 On many version control systems with atomic multi-change commits, a change list (or CL), change set, update, or patch identifies the set of changes made in a single commit. This can also represent a sequential view of the source code, allowing the examination of source "as of" any changelist ID.
* Checkout 🡪 To check out (or co) is to create a local working copy from the repository. A user may specify a specific revision or obtain the latest. The term 'checkout' can also be used as a noun to describe the working copy. When a file has been checked out from a shared file server, it cannot be edited by other users. Think of it like a hotel, when you check out, you no longer have access to its amenities.
* Promote 🡪 The act of copying file content from a less controlled location into a more controlled location. For example, from a user's workspace into a repository, or from a stream to its parent.
* Pull, push 🡪 Copy revisions from one repository into another. Pull is initiated by the receiving repository, while push is initiated by the source. Fetch is sometimes used as a synonym for pull, or to mean a pull followed by an update.
* Repository 🡪 The repository is where files' current and historical data are stored, often on a server. Sometimes also called a depot.
* Revision 🡪 Also version: A version is any change in form.
* Tag 🡪 A tag or label refers to an important snapshot in time, consistent across many files. These files at that point may all be tagged with a user-friendly, meaningful name or revision number. See baselines, labels and tags.
* Trunk 🡪 The unique line of development that is not a branch (sometimes also called Baseline, Mainline or Master)
* Working copy 🡪 The working copy is the local copy of files from a repository, at a specific time or revision. All work done to the files in a repository is initially done on a working copy, hence the name. Conceptually, it is a sandbox.





The development will happen on Daily/Weekly/Monthly/need basis. Depending upon the methodologies they follow the development approach will differ from project to project. One such method is parallel development and parallel releases. In this parallel development approach multiple releases will be scoped, and development will be going in parallel by creating multiple branches/tags using version control tools.

Once the developer does any code development, then the same changes needs to be moved to all the future branches which are scoped after this release. But with the increased number of branches and dynamic code propagation paths (which changes after every Go-Live), this will become a challenge for the developers to identify the right code propagation path at that point in time and check-in their changes to multiple branches. This possess a great risk of missing check-ins in the required branches and unwanted check-ins in the lower release branches which in turn puts the Production environment at stake after the release Go-Live.

To overcome the above situations there is a great need of synching up all the code changes from the lower release branches to higher release branches.  But with the increased number of branches and dynamic code propagation paths (which changes after every Go-Live), this is becoming a challenge for the developers to identify the right code propagation path at that point in time and check-in their changes to multiple branches. This possess a great risk of missing check-ins in the required branches and unwanted check-ins in the lower release branches.

# Approach

In the market there are tools like git, perforce... etc. which has in-built mechanism of handling auto code merge. But when coming to Collabnet Subversion there are limitations to it. To address the above situation, a new utility was developed which is generic and can be used on any Collabnet Subversion.

## Auto Code Commit

To achieve the “Auto Code Commit”, a utility was developed with the help of SVN hooks, custom scripts and config files, which lets the developers to move their code into lower branches automatically without developer’s intervention. Developers will only be needed to intervene manually in the event of a code conflict. Check-in order of code commit which is custom controlled through a property file.

## Process

To propagate the code automatically to all the higher branches, we have developed a tool which will detect code changes on a branch/tag in a repository and propagates those changes to all the higher branches as the per the code propagation path which will be specified in a config file.

To have achieve Auto Code Commit, we have used below stack:

* Shell Scripting
* SVN commands
* SVN hooks

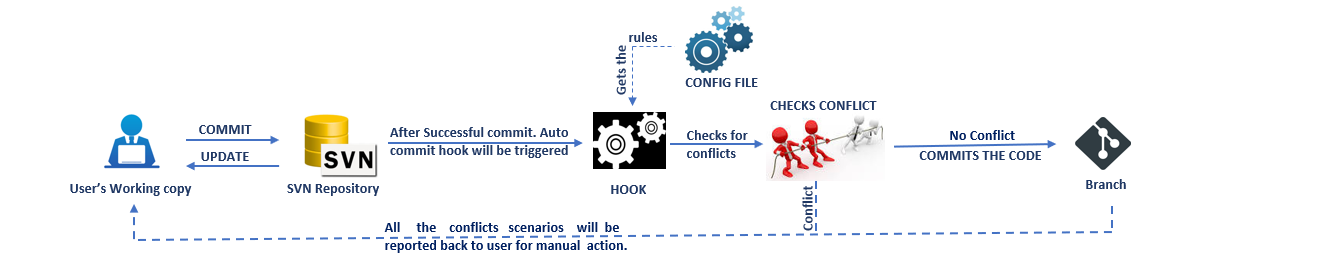
This can be only enabled at the repository level and all the changes done on branches/tags under the that repository will be monitored. Even though it was enabled at the repository level, we can exclude and include branches/tags using the config file.

Once the Auto Code Commit is enabled on any of the repository, the code committed under that repository branches will be automatically monitored/detected and merged into the respective higher branches (only non-conflict scenario). Auto code commit propagation paths can be modified using a config file as per the needs. Code changes will have merged based upon on the revision number. When there is conflict observed at the time of auto commit process, auto commit will print all conflict messages on the SVN client pop window (Tortoise SVN). If there is any conflict observed as part of a revision, whole revision will be ignored. No partial commits will happen.

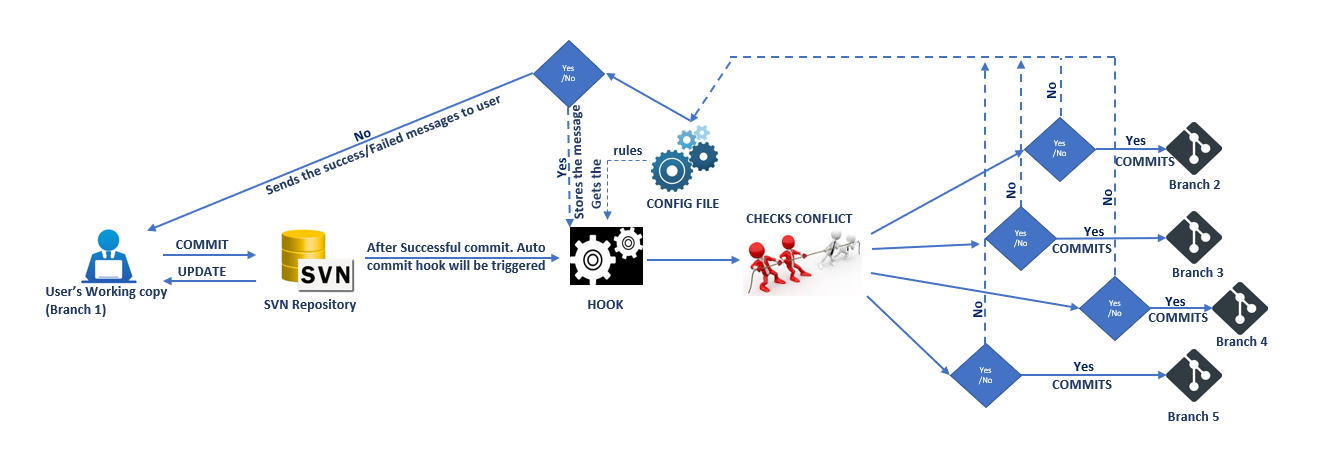
This utility is generic and can be used across all the repositories under CSVN and this can be used in any project which is using CSVN as a version control tool.

## Please find the flow diagram below

1. **Branch/Tag to Branch/Tag Flow Diagram:**

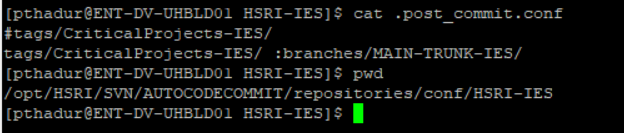


1. **Branch/Tag to multi Branch/Tag Flow Diagram:**



## Understanding config file

We have one “.post\_commit.conf” for every repository level. With the help of this config file we will configure all the branch/tag names which all needs to be enabled and their corresponding propagation branch/tag names. Please find the below screenshot of “.post\_commit.conf” file for reference:





In the first line, we will declare all the branch names that needs to be monitored by the post commit hook followed by the “#” symbol. As per the above screenshot, auto commit is enabled on WBT IES branch only.

From Second line we mention all the branch name and all its propagation branch/tag names followed by “:” symbol. As per the above screenshot, whatever the commits detected on WBT IES will be auto propagated to MAIN-TRUNK-IES branch.

## Important locations and files

Please find the below important locations and config file locations:

**Checkout location:** /opt/SVN/AUTOCODECOMMIT/repositories/

**Config file location**: /opt/SVN/AUTOCODECOMMIT/repositories/config/<Repository\_Name>

**Config File Name:** post\_commit.conf

**Log location:** /opt /SVN/AUTOCODECOMMIT/logs/<Repository\_Name>/<Current\_Date>

**Log Name Format:** <branch/tag><Branch\_Name>\_POSTCOMMIT.log

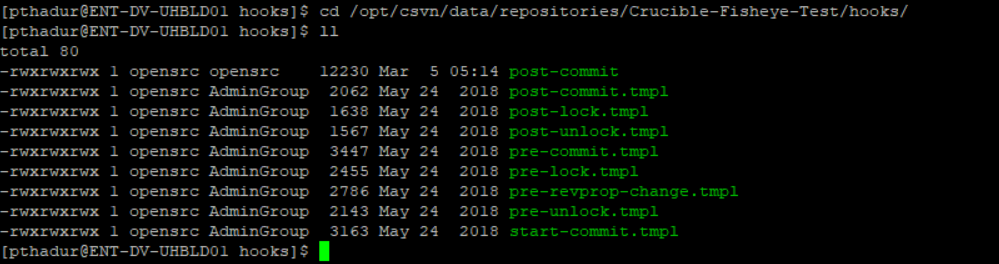
## Enabling and disabling Auto Commit

If any time if we want to disable the “Auto Code Commit”, we need to go that repository hook folder and rename the post-commit file to post-commit.tmpl.

**Post commit file location:** /opt/csvn/data/repositories/<Repository\_Name>/hooks

**Ex:** /opt/csvn/data/repositories/HSRI-IES/hooks

Please find the below screenshot for reference:





# Developer's Stand point

From the developer's prospective, everything remains same and unchanged. All they need to do is to commit the code changes as per process and once the code commit is successful the “Auto Code Commit” hook will be triggered automatically and merges the code into the respective branches.

**Note:**

Please read SVN Tortoise client pop messages carefully before clicking ‘OK’. If the message appears in RED color, then please follow the message and resolve the conflicts manually on the respective branches/tags and commit the changes manually in higher to lower branches/tags.

Please make sure pop message option is enabled on your Subversion client software.

# POC and implementation

We have done the POC and implemented it on one of the public sector projects handling the Health Care application. This project follows Hybrid model and is using Collabnet Subversion as a source code repository. As a part of the project runway development scope, multiple parallel releases are planned like Hotfix, Weekly, Bi-Weekly, Monthly releases long with the parallel development. There are approximately 200 developers and 100 testers who are working on different release. After every release Go-live or on priority basis every commit/change done by the respective developer should be propagates to all the future release branches. It takes ~3 hours for each developer to propagate all the changes to all the future release branches.

After implementing this, every commit done by a developer is detected automatically and propagated to all the future release branches if there are no conflicts and conflicted scenarios will be showed to the developer on a pop window asking them to resolve the conflict and commit on the respective branches. All this process will take only 10-20 sec (may vary depending upon no of files) after a successful commit on the source branch.

# Pre-Requisites

1. To enable Auto commit, the entire code of all the required branches must be checked out on to the server. To accommodate this, we would require space on server.

# Features

1. All the files/file types which are mentioned in Ignore list are not part of Auto commit process. Hence, they must be manually committing to all the branches in propagation path.
2. We can modify Ignore list at any point of time.
3. It is very easy to Enable and disabling the “Auto Code Commit”.
4. “Auto Code Commit” utility can be used on across multiple repositories without any modifications.
5. “Auto Code Commit” utility can be used on any CSVN.
6. We can ignore any commit at any point of time by just a passing pre-defined custom message as a part of commit message.
7. Auto Commit can be enabled at repository level and all the branches/Tags falls under that repository can be controlled.
8. "AUTO COMMIT" is the key word that is used to detect the auto commits attempted by AUTO COMMIT. Please refrain using this key word in any of the commit messages.

# Best Practices

**Do's:**

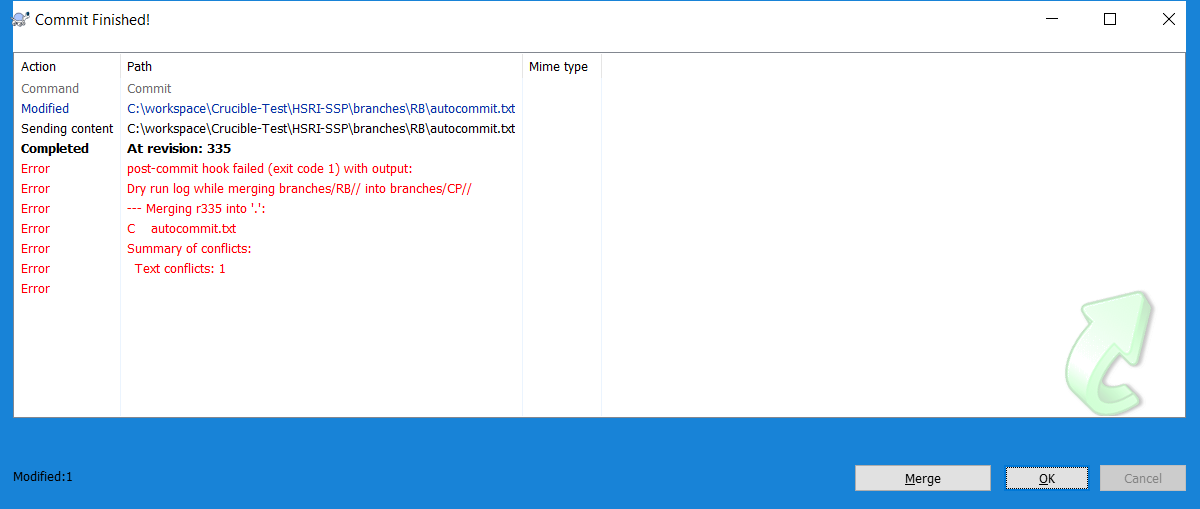
1. Any SVN client pop-up with RED color must be read and understood carefully. DO-NOT close the pop-up until it is preserved for future reference.
2. In case of conflict scenarios, resolve all the mentioned conflicts and manually commit them in reverse propagation path.

**Don’ts:**

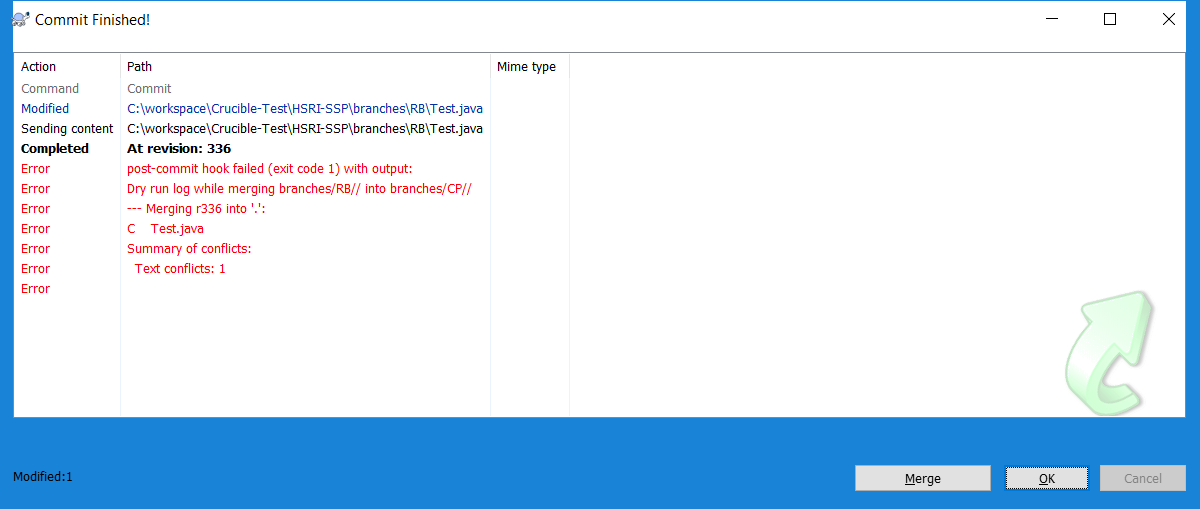
1. ‘Commit’ must not contain files from multiple Branches/tags. E.g... Branch1 & Branch2 files must not be committed in a single attempt.
2. Any ‘Commit’ containing at least one file from the **Ignore List,** entire ‘Commit’ is **ignored for “Auto Code Commit”**. However, it will be committed at the Source branch. Hence developer should commit any such ‘Changeset’ manually to entire propagation path.
3. **DO NOT** click on **'cancel'** button when the commit is in progress. As this will stop only SVN client but not the background Auto Commit Process.

**Conflict commit message (Tortoise SVN client):**

**Scenario 1 *(Tortoise SVN client)*:**

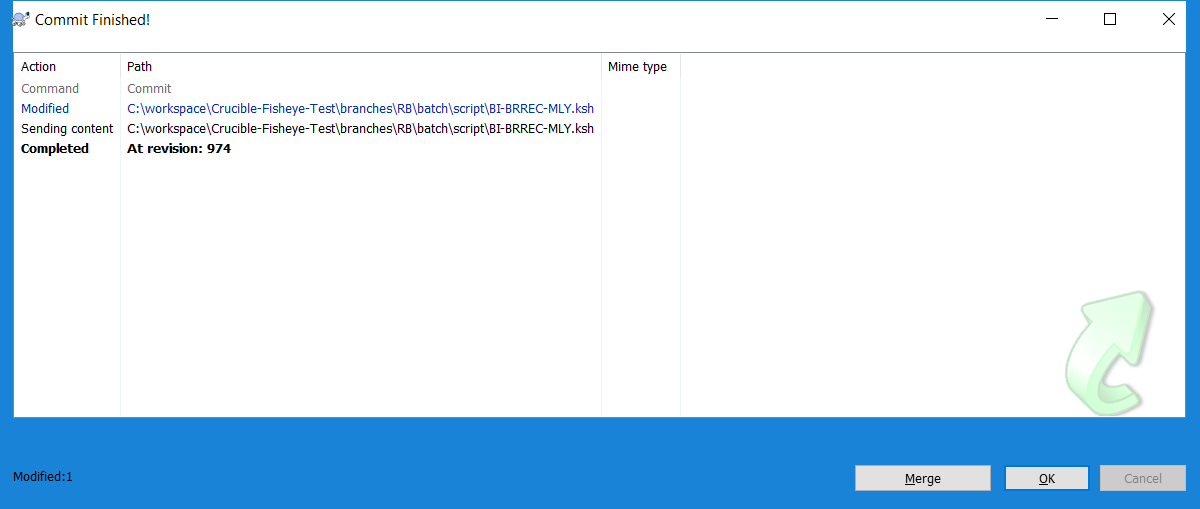


**Scenario 2 *(Tortoise SVN client)*:**



If you see a regular SVN client pop with a revision number in it and nothing else, then all the changes have been propagated into higher branches successfully.

***Clean commit will appear like below (Tortoise SVN client):***



# Limitations

1. Any Add/Delete operations cannot be processed under this “Auto Code Commit” utility.
2. Weekly maintenance is required on all the checkout locations.

# Summary

To summarize, the present DevOps world demands utmost flexibility in terms of the number of code check-ins and the daily code deployments. This increases, huge stress and manual effort on the developers especially when Subversion is the version control system. To ease the entire process and to pave the way for continuous builds and deployments (CI/CD), SVN Auto Commit Hook opens the door with latest automation trends.

After successful implementation of this on the live project, code changes done on the source branch has been detected dynamically and then propagated those changes to all the future release branches in all the non-conflict scenarios which helped developers in saving their time. This also helped in reducing the manual mistakes while propagating the changes to the higher release branches and code miss scenarios have been reduced to zero.