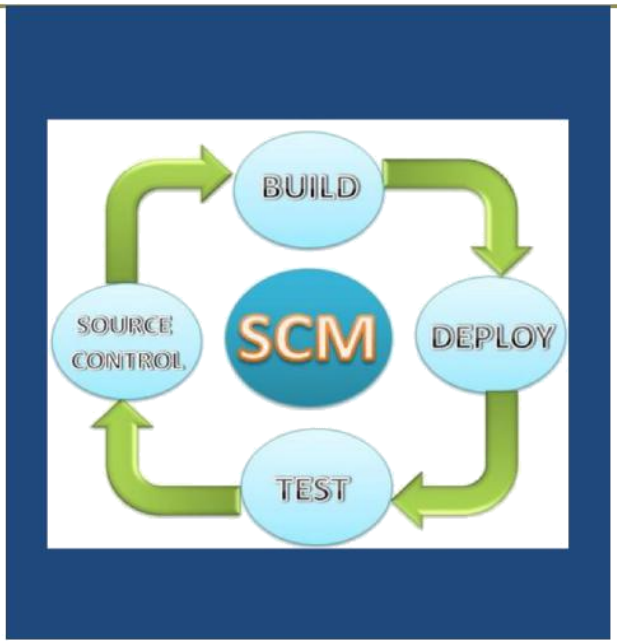


SCM/Build & Release Management

Nagababu Kanneganti

Contents

Software Configuration Management

Software Configuration Mgmt is a software development support and control function.

Specializing in software building, installation packaging, change management and configuration management.

SCM ensures the integrity, reliability and reproducibility of software from conception to release to retirement.

Typical SCM Functions

Software Builds

Source Code Administration

Install Packaging

Software Change Management

Configuration Management

Software Configuration Management Plan Purpose

The purpose of this plan is to provide basic guidelines for how source code and software builds will be managed.

This document will cover source code administration, build environment standards and define the process by which new components will be added to builds, and a common understanding of how to manage broken builds.

The intent of Software Configuration Management Plan is to ensure consistent software Quality across projects by ensuring that:

- All components are managed in a source control tool
- All components are compiled on an independent and controlled build machine
- Build machine specifications are controlled and documented

Typical SCM Plan Requirements

- Well Defined Set of Configuration Items to be Managed

- Assign Roles and Responsibilities
- Define Software Configuration Management Standards and Procedures

Source Code Management / Version Control System:

Source Code is typically a file, or set of files, that contain human readable text that when it is interpreted or compiled by a computer, it performs a task, or runs a computer program. The Internet browser you are using to read this web page is an example of a compiled computer program.

This file, or set of files, contain a set of instruction that tell the computer what to do. Most computer programs are made up of hundreds or thousands of files containing source code.

These computer instructions are written in a computer language that can be interpreted or compiled on the computer by tools known as an Interpreter or Compiler.

Examples of computer languages are C, C++, Java, Python, Perl and C#.

Version control is defined as series of processes, used to manage and maintain multiple revisions of all the objects in a shared repository.

With version control, we can

- Keep track of changes and process control
- Investigate changes and revert
- Several people can work on same project or file simultaneously
- Merge changes
- Provides audit report who modified, when modified, what are the
- Changes made etc…
- Possible to recreate the source base from which software system was built

Advantages of VCS

- A place to store your code
- Historical record of what was done over time
- Synchronization between the developers
- Facilitates automated build and deploy
- Enables the developers to work in parallel.
- Standardized application lay out.

Without VCS what are all disadvantages:

- Needs to maintain the source code in a shared directory.
- Every one overwrites the source code.
- No revert
- Not feasible to work in parallel.
- No process control, no change control.
- Not possible to get who modified the file, what are the changes he made etc…
- Not possible to resolve merger conflicts.

Example tools for VCS:

Commercial tools:

Perforce, IBM clearcase, VFS, TFS,

Open Source tools:

Subversion (SVN), CVS, Git, Mercurial ..etc

What is a Repository?

Repository is much like an ordinary file server except that it remembers everything that is done to your files and directories.
Network-wide file system resource which stores version-controlled data. We can store all source code and project related documents such Project Plan, Design Document, DSRS, etc.

- Version tracking
- Collaboration and sharing files
- Historical information
- Retrieve past versions
- Manage branches

Subversion (SVN):

Subversion is a version control system, successor to CVS

- Manages files and directories over time
- Maintains files and directories in a
- central repository
- Can manage any collection of files,
- including source code
- Also known as SVN
- Platform independent

History

From a CollabNet perspective, the SVN project started in 2000. After encountering

numerous problems with integrating their flagship product CollabNet EnterPrise

Edition with CVS, they decided to create a new open source tool that fixed the CVS

limitations.

Hence, Subversion was born...

SVN Repository: The repository is where files are stored under Subversion on the server.

Working Copy: The contents of a project in the repository are usually copied into a local directory. This is known as the working directory, and is separate from the repository

Checkout & Check-in: The operation of copying the project to the local directory is called "check out" and the reverse is "check in".

Repository Vs Working copy:

- Project code is stored in a server in a data store referred to as a "repository."
- Developers "check out" copies of the project code into their local environments. These
- Copies are referred to as "working copies." After making changes to a working copy, the
- Developer "commits" changes to the repository.
- Other developers get these changes by "updating" their working copies.

Revision number:

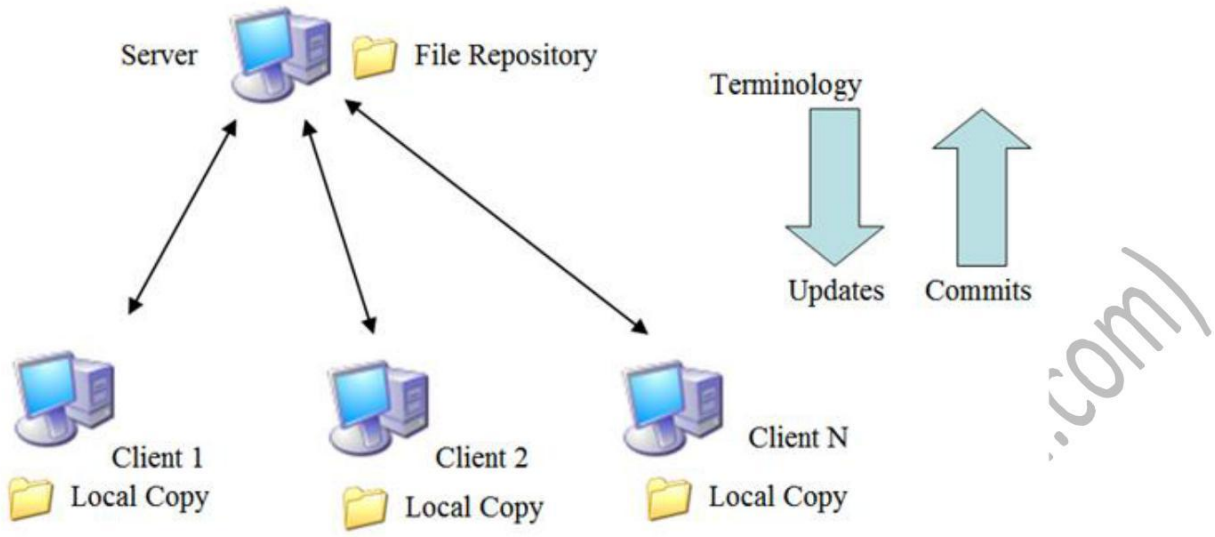
- Global, Snapshot of the repository, unique id.
- Revision numbers are global across the whole repository. Each commit increases the version of the repository
- Identify how the entire repository looks at that instant in time. A commit creates a snapshot of the entire tree in the repository at that revision number
- After committing to a repository with revision number n, the repository is changed to version n+1.
- After the repository is initially created, it is an emptyfolder and has revision number 0.

Atomic commits:

- A collection of modifications either goes into the repository completely, or not at all.
- In other words, a commit will either altogether succeed, or it will altogether fail.
- Data Integrity
- Exception – limits on http.

Subversion Architecture:

Subversion is client-server based disconnected architecture.



Installation:

Require Subversion Installer and Apache server as subversion independently does not support remote access and security

Apache download:

http://httpd.apache.org/download.cgi (Apache 2.2)

Subversion download:

http://subversion.apache.org/packages.html Version: version 1.7.6 (r1370777)

Configuring Subversion and Apache :

1. Install subversion and Apache

2. Create a Repository using svnadmin

command svnadmin create D:\MyProject

3. SVN & Apache configuration:

a) Copy C:\Program Files\Subversion\bin\mod_authz_svn.so and C:\Program Files\Subversion\bin\mod_dav_svn.so from subversion instal path to Apache (C:\Program Files\Apache Software Foundation\Apache2.2\modules)

b) Open C:\Program Files\Apache Software Foundation\Apache2.2\conf/httpd

Add below 3 lines at the end of the Load modules

LoadModule dav_module modules/mod_dav.so

LoadModule dav_svn_module modules/mod_dav_svn.so

LoadModule authz_svn_module modules/mod_authz_svn.so

c) At the end of the file (httpd file) add the below lines <Location /svn>
DAV svn SVNPath D:/MyProject

</Location>

4) Restart Apache

Subversion basic commands:

How to create a workspace ?

svn checkout http://localhost/svn How to add files to workspace? Enter into the workspace directory

svn add filename → Adding single file

svn add –R *.* → Adding multiple files at a time

How do we send the modified files to svn repository server?

svn commit –m file.txt "adding files to sever" → adding single file

svn commit –m "adding files to sever" → by default commit pickups all changed files and send to repository server

All basic svn commands:

svn checkout/co

svn checkout or svn co. This command is used to pull an SVN tree such as http://localhost/svn from the server. You should only need to do this once. If the directory structure is changed (as is sometimes necessary), you may occasionally need to delete your local sand box and re-check it out..

svn add

svn add. When you are creating a new file or directory, you need to tell the SVN server about it. This command does that. Note that the file won't appear in the repository until you do an svn commit (see below).

svn delete (del,rem,remove)

svn delete. **This** does what it says! When you do an **svn commit** the file will be deleted from your local sand box immediately as well as from the repository after committing.

svn status (st, stat)

svn status. **This** command prints the status of working directories and files. If you have made local changes, it'll show your locally modified items.

svn update/up

svn update or svn up. This command syncs your local sand box with the server. If you have made local changes, it will try and merge any changes on the server with your changes *on your machine*.

svn commit/ci

svn commit or **svn ci**. This command recursively sends your changes to the SVN server. It will commit changed files, added files, and deleted files. Note that you can commit a change to an individual file or changes to files in a specific directory path by adding the name of the file/directory to the end of the command. The -m option should always be used to pass a log message to the command. Please don't use empty log messages (see later in this document the policy which governs the log messages).

svn diff (di)

svn diff. **This** is useful for two different purposes. To do this, simply edit the files in your local sand box then run **svn diff > FILE.patch** from the root of your BLFS directory. The second use is to find out what has changed between two revisions using: **svn diff -r revision1:revision2 FILENAME**. For example: **svn diff -r 168:169 index.xml** will output a diff showing the changes between revisions 168 and 169 of index.xml.

svn move

svn move SRC DEST or **svn mv SRC DEST** or **svn rename SRC DEST** or **svn ren SRC DEST**. This command moves a file from one directory to another or renames a file. The file will be moved on your local sand box immediately as well as on the repository after committing.

SVN Cat:

svn cat http://localhost/svn/test.txt shows the contents of a file on console.

SVN Cleanup:

Svn cleanup will cleans up if any conflicts happens on SVN workspace

SVN export:

Export a clean directory tree, which is not a workspace as we cannot perform any svn operations on exported codebase.

svn export http://localhost/svn
Export complete.

SVN import

Svn import pushes the non workspace files to svn repository server

svn import file.txt http://localhost/svn/file.txt -m "imported file.txt tory"

SVN info

Which is a workspace command and provides information about SVN workspace URL, revision no etc.

SVN list

Lists the files and directories inside a repository locations

svn list http://localhost/svn

SVN log

Displays the history of a file or a directory in SVN with all it previous revision

numbers. svn log http://localhot/svn

SVN mkdir

Creates the directory inside workspace, no need to add a directory by creating on file system.

SVN Directory structure

Depending on type release process the SVN structure will be decided. The conventional SVN structure is shown below

svn\trunk,

svn\branches,

svn\tags

svn copy

We can create the branches and tags using svn copy comand

svn copy <source branch url> <destination branch url> -m "comment"

svn switch

This command is used to switch between the branches without checking out the codebase

svn switch http://localhost/branches/branch1

svn blame:

svn blame — Show author and revision information inline for the specified files or URLs.

svn blame http://localhost/svn/test.txt

svn merge This comand used to merge code from one branch to other branch

svn merge http://localhost/release2

Release Management:

Release management is a software engineering process intended to oversee the development, testing, deployment and support of software releases. The practice of release management combines the general business emphasis of traditional project management with a detailed technical knowledge of the systems development lifecycle (SDLC) .

Release management usually begins in the development cycle with requests for changes or new features. If the request is approved, the new release is planned and designed. The new design enters the testing or quality assurance phase, in which the release is built, reviewed, tested and tweaked until it is ultimately accepted as a release candidate. The release then enters the deployment phase, where it is implemented and made available. Once deployed, the release enters a support phase, where bug reports and other issues are collected; this leads to new requests for changes, and the cycle starts all over again.

An Agile Approach to Release Management

Teams practicing Agile Software Development value working software over other artifacts. A feature from the release plan is not complete until you can demonstrate it to your customer, ideally in a shippable state. Agile teams strive to have a working system ("potentially shippable") ready at the end of each iteration. Thus Release Management should be easy for an ideal agile team, as agile teams, in theory, are ready to release at regular intervals, and the release management aspect is the customer saying "ship it!."

Agile teams work under the same constraints as other software development teams, having to address issues of maintenance and support, the need for external artifacts like documentation and training, and the reality that not every customer can accept change at the rate that an agile team can (in theory) deliver it. To attain the goal of a shippable product at the end of every iteration, an agile team must manage the flow of change from a customer, and maintain high discipline and good engineering practices.

Please refer the mentioned URL for detailed agile process

http://www.scmpatterns.com/pubs/crossroads-mirror/2008-05-CMCrossroads.pdf

SVN Administration

SVN Administration involves below activities

- Creating repository
- Creating authentication and authorization repository
- Provide user credentials to the developers
- Giving access to users on respective repository locations
- Maintaining groups in svn access file
- Creating branches for new releases
- Help the developers while merging the code
- Take the dump of SVN repository on frequent intervals

SVN Authorization setup

Create authentication file on SVN server using apache htpasswd file

Create D:\etc\svn_auth_file by using below command inside apache bin

cd C:\Program Files (x86)\Apache Software

Foundation\Apache2.2\bin> htpasswd.exe -c D:\etc\svn-access harry

passwd: *****

Reconfirm: *****

Create authorization file to givie the proper access to the users created

Create another file svn-access

Provide access to the users which you created

[groups]

devtemaf1 = seetaram, surech, Janardhan, santosh

[svnrepo:/branches/Feature1]

harry = rw

[svnrepo:/branches/Feature2]

sally = rw

Configure the above file in httpd.conf file where your repository added

<Location /svnrepo>

DAV svn

SVNPath D:/SVN_REPOSITORY/Project1

AuthType Basic

AuthName "Subversion Project1 repository"

AuthUserFile D:/etc/svn-users

Require valid-user

AuthzSVNAccessFile D:/etc/svn-access

</Location>

SVN Commands

These commands works when you configure authorization on SVN

svn lock : which creates a lock on single file, so that no other developer can change this file

but still other can unlock this file and change it with below command

svn unlock

svn lock a.txt

svn unlock --force a.txt

SVN Hooks

A hook is a program triggered by some repository event, such as the creation of a new revision or the modification of an unversioned property.

A SVN hook resides inside the SVN repository, which you create after SVN installation.

Types of hooks

Pre-commit:

Notification just prior to commit completion

The pre-commit hook is run just before a commit transaction is promoted to a new revision. Typically, this hook is used to protect against commits that are disallowed due to content or location (e.g., your site might require that all commits to a certain branch include a ticket number from the bug tracker, or that the incoming log message is nonempty)

Example for pre-commit hook

Requirement: Users should be restricted if they commit anything without log message

Write a batch script to implement the pre-commit hook in windows.

```
@echo off
 :: Stops commits that have empty log messages.
 @echo off
 setlocal
 rem Subversion sends through the path to the repository and transaction id
 set REPOS=%1
 set TXN=%2

 svnlook log %REPOS% -t %TXN% | findstr . > nul
 if %errorlevel% gtr 0 (goto err) else exit 0

 :err
```

```
echo. 1>&2
echo Your commit has been blocked because you didn't enter a comment. 1>&2
echo Write a log message describing the changes made and try again. 1>&2
echo Thanks 1>&2
exit 1
```

Post-commit

post-commit — Notification of a successful commit.

The post-commit hook is run after the transaction is committed and a new revision is created. Most people use this hook to send out descriptive emails about the commit or to notify some other tool (such as an issue tracker) that a commit has happened. Some configurations also use this hook to trigger backup processes.

If the post-commit hook returns a nonzero exit status, the commit will not be aborted since it has already completed. However, anything that the hook printed to stderr will be marshalled back to the client, making it easier to diagnose hook failures.

SVN UI

All the SVN activities can be done from user end using tortoise svn which is a graphical user interface for end users.

TortoiseSVN is an Apache™ Subversion (SVN)® client, implemented as a windows shell extension. It's intuitive and easy to use, since it doesn't require the Subversion command line client to run. Simply the coolest Interface to (Sub) Version Control!

You can download and install the Tortoise SVN from below URL

http://tortoisesvn.net/downloads.html

Features:

- Easy to use
- All commands are available directly from the windows explorer.
- Only commands that make sense for the selected file/folder are shown. You won't see any commands that you can't use in your situation.
- See the status of your files directly in the Windows explorer
- descriptive dialogs, constantly improved due to user feedback
- allows moving files by right-dragging them in the windows explorer