

# SVN

# TEK LEADS

## **Table of contents**

1. What is Version Control
2. Why we need Version control
3. SVN Terminology
4. SVN Installation (Server & Client)
5. Workflow
6. Initial Folder Structure
7. Operation we can perform with SVN
8. Checkout Process
9. Adding Files
10. Check-in process
11. Update operation
12. Files Revision History
13. Checking File Differences
14. Deleting Repository files
15. Show Logs
16. Locking
17. Revert Process
18. Working with Branches
19. Cutting Branches from Trunk
20. Subclipse plug-in with Eclipse

**What is Version Control?**

A version control is a system (also known as a Revision Control System). It maintains repository of files with monitored access. Every change made to the file is tracked, along with who made the change, why they made it and when they made it.

**Version tracking**

If you are a programmer or web designer and want to keep every version of java program or an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. Since version control systems keep track of every version of the software, this becomes a straightforward task. Knowing the what, who, and when of changes will help with comparing the performance of particular versions, working out when bugs were introduced (or fixed), and so on. Any problems that arose from a change can then be followed up by an examination of who made the change and the reasons they gave for making the change.

**Advantages of Version tracking**

It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

Without a tool like Version Control project development would get horrible. Rather than focusing on actual project functionality, team would put more efforts and time on maintaining the stability of the code base.

You always need to work w.r.t the latest code so there is a need of a central system which tracks the latest code at any point of time.

- We need a system which can assure us that the code never gets override by other developer's code.
- Manually monitoring this central system is practically never possible.
- We need a tool which can address this concern at any given point of time

**Open Source Version Control tools**

- ❖ cvs (<http://www.nongnu.org/cvs/>)
- ❖ Sub Version (<http://subversion.apache.org/>)
- ❖ GitHub (<http://git-scm.com/>)

As part of this tutorial, let's understand what Subversion is and how to use it

**SVN**

- ❖ Subversion (SVN) is a source code Repository Server
- ❖ SVN is a free/open-source version control system
- ❖ SCM (Software Configuration Management) implementation.
- ❖ It allows to track changes in files and directories.
- ❖ It is centralized(One Server)

### **Terminology of SVN**

- ❖ The Repository
- ❖ Revision
- ❖ The Working copy
- ❖ Updating and Conflict Resolving
- ❖ Merging
- ❖ Commits

### **Installation**

In order to work with SVN we need SVN server software and SVN client.

- SVN server would be installed in the machine where central code and documents would be maintained. This helps getting a controlled access for getting data inside and outside the central place.
- SVN clients are installed in developer's machine and are the means to perform operations with the code/files available inside SVN server.

### **SVN Server**

There are multiple SVN server software available in the market and **VisualSVN** is one of such kind quite commonly used during project development.

VisualSVN server maintains code/files related to entire projects that run within your org.

These projects would be managed as separate repositories, where each repository would be pointing to a single project.

It provides secured access to employees for accessing this codebase.

### **Working with Visual SVN**

Download: <https://www.visualsvn.com/server/download/>

Features:

- Allows you to create repositories which can be exposed as HTTP urls (e.g.: `https://<machine_name>:8678/svn/<project_name>/`)
- Allows you to add users.
- Allows you to organize users in groups.
- These individual users or groups could be given access to created repositories.
- Only those who are permissions, can access the repositories.
- Maintains history of changes done by all users.

### **SVN Client**

This is a software which would be installed in client machines. (In your laptops / desktops)

Using this you can connect to the repository created in SVN Server

There are multiple clients available in market and the developer has the freedom to choose the tool they are comfortable to use. (Within one team multiple members can work with different tools)

This tool allows to perform all operations w.r.t the repository managed by SVN server.

Egs: EclipsePlugin, TortiseSVN

### Working with Tortoise SVN

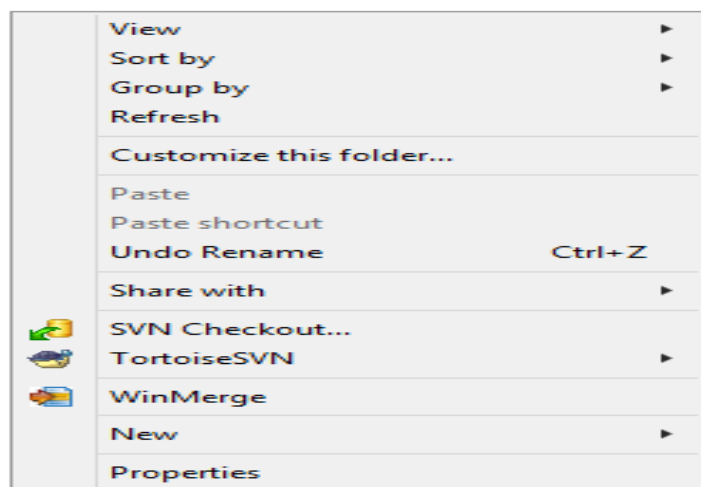
Download: <http://tortoisesvn.net/downloads.html>

Features:

- This is installed in client machine and helps in connecting to SVN Server software
- You can add /update/modify code in the repository.
- You can check history of changes done on the repository.
- Allows you to navigate through repository.
- Allows you to work and manages branches.
- Allows to reset the code in repository to any revision.

Once Installation is completed, re-start your computer.

Then verify Installed Software's by right-clicking on mouse. You can see the below options.



### End To End Process

- Project Manager (PM) drops mail to SVN Admin team to create new repository for the project.
- Manager would also communicate SVN IDs of those candidates who should be given access to repo.
- In case new employees join the org, then first SVN ids would be generated for the employee, and then depending on the project, permission would be given to the candidates.
- SVN admin creates repository, grants permissions to appropriate users and shares the SVN URL for the repository with project manager.
- This SVN URL would be shared with entire team.
- Install TortoiseSVN in your local machine.
- Decide on a folder structure in your local machine which you want to make in sync with the repository. That means if you want to work with repo, then you should do all modifications in your local machine and send your changes to the repo using TortoiseSVN tool.
- Using the SVN URL shared with you, check out the entire project code in your local machine.
- Bind the checked out code in your eclipse(if is maven, then simply do maven import in eclipse)
- Do all code modifications and sync your changes to the repository.
- Any change that you put in repository (even space change) would be tracked with your SVN ID. So be very particular when committing something to repository because everyone can identify who is doing what in the team.

**Things you need to do after joining company or getting allowed to do project**

- Know the SVN server URL of your project
- Ask you manager for getting your SVN ID generated and get permissions for accessing the project code/documents
- Install TortiseSVN in your local machines
- Get going with the process

**Folder Structure of Repositories****1) Trunk****2) Branches****3) Tags**

**Trunk:** The trunk is a directory where all the project code lies. Code from trunk would be later on shipped to production environment. It's important that trunk always stays stable (no compilation error or no functional issues).

**Branches:** Whenever a new major functionality is being developed, initially the code might not be stable. And is not in a shape to be placed in trunk. But in parallel we cannot risk keeping the code in individual machine. So we need to commit it to repo. This can be committed inside branches. Branches are cut from the main trunk and contains entire code on trunk from the revision the branch is cut from

**Tags:** Stable versions of trunk is cut as tags. Generally these are derived to maintain a stable working functional code always. This might be helpful for providing demos at any given point of time while the project development is happening.

**Operations that are expected to perform while working with SVN**

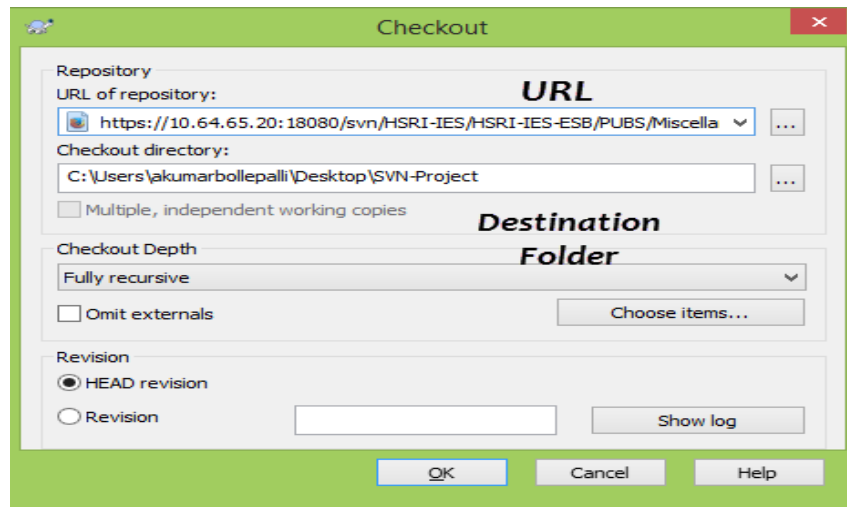
- ❖ Checkout
- ❖ Add
- ❖ Check-in
- ❖ Update
- ❖ Diff
- ❖ Delete
- ❖ Show Logs
- ❖ Lock
- ❖ Revert
- ❖ Reverting a file to older revision
- ❖ Getting SVN URL
- ❖ Cutting Branch from trunk
- ❖ Switching branches
- ❖ Merging branches back to trunk

**SVN Checkout**

For getting code from SVN server to your local machine for the first time, use "SVN checkout" operation.  
Process:

- Go to Empty folder which you want to bring in sync with the repository.
- Right Click -> SVN Checkout

- On checkout popup, provide the SVN URL from where you want to get code in your local machine. Generally the root location of the project code is chosen.
- Eg: <https://168.456.0.45:8443/svn/AshokSoft/trunk>

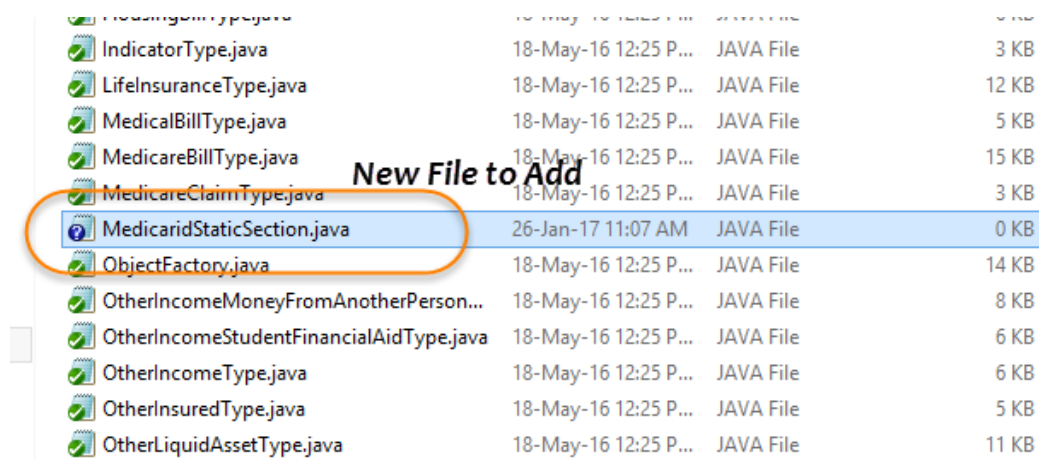


### SVN Add

On creating a new file, its state would be un versioned. When you “Add”, you ensure that when you would commit for the next time you would pick up this file for commit.

#### **Process:**

- Right Click for newly created file -> “SVN Add”
- This add symbol (blue colored question mark) over your file.
- If you add something and change your mind before committing, you can unscheduled the addition using “SVN Revert operation”.



### Commit (Check – In)

The act of placing your modified files or newly added files inside the repository is called commit operation.

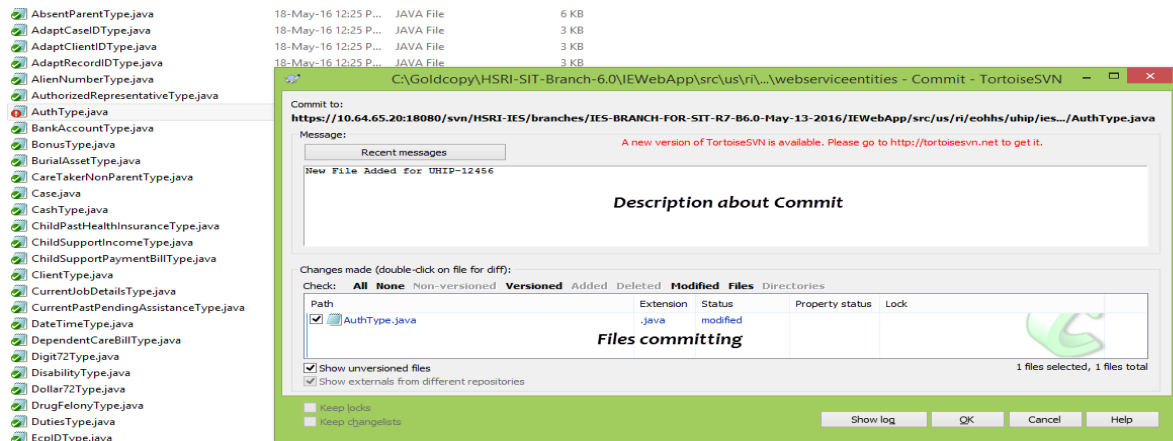
First you have to create/modify a file locally and then you can commit it.

While committing you also need to give valuable meaningful information about what changes you are actually committing.

Be very careful while commit, and make a habit to always check the “SVN Diff” before commit so you don’t put unnecessary changes in repo.

#### Process:

- Right Click on file you want to send to Repo -> SVN Commit
- In the popup that comes up, double click the file to check the difference
- Once changes are confirmed, add appropriate description for your commit and click “OK”



#### Update

Getting the latest code from the repository in your local machine is called as update operation.

You should always work w.r.t latest SVN copy, thus you need to always keep on updating your local machine code to the repo code.

Every day morning, first thing that you do is update your local machine to latest repo code.

Also while committing your make a habit to update before.

You might have some local modifications to a file and it is possible that someone else might have modified the same file and committed in the repo.

Thus on updating that there are 2 possibilities













- Either the tool would automatically merge the latest repo changes with your local working copy
- Or tool would leave you in a conflicted state. This should be further manually resolved using “Edit Conflict” option of tortoise SVN

On trying to commit a code, if your changes are not w.r.t latest repo code, then the tool would block your commit and request you to update your code first.

#### Process:

- Right on the file / folder that you want to update to the latest repo code -> “SVN Update”
- Prefer taking an update on the root folder so all your files under that gets updated



	AdaptClientIDType.java	18-May-16 12:25 P...	JAVA File	3 KB
	AdaptRecordIDType.java	18-May-16 12:25 P...	JAVA File	3 KB
	AlienNumberType.java	18-May-16 12:25 P...	JAVA File	3 KB
	AuthorizedRepresentativeType.java	18-May-16 12:25 P...	JAVA File	17 KB
	AuthType.java	26-Jan-17 11:05 AM	JAVA File	2 KB
	BankAccountType.java	18-May-16 12:25 P...	JAVA File	14 KB
	BonusType.java	18-May-16 12:25 P...	JAVA File	2 KB
	BurialAssetType.java	18-May-16 12:25 P...	JAVA File	10 KB
	CareTakerNonParentType.java	18-May-16 12:25 P...	JAVA File	3 KB
	Case.java	18-May-16 12:25 P...	JAVA File	50 KB
	CashType.java	18-May-16 12:25 P...	JAVA File	3 KB
	CashType.java	18-May-16 12:25 P...	JAVA File	3 KB

**Modified File, Need to Commit**

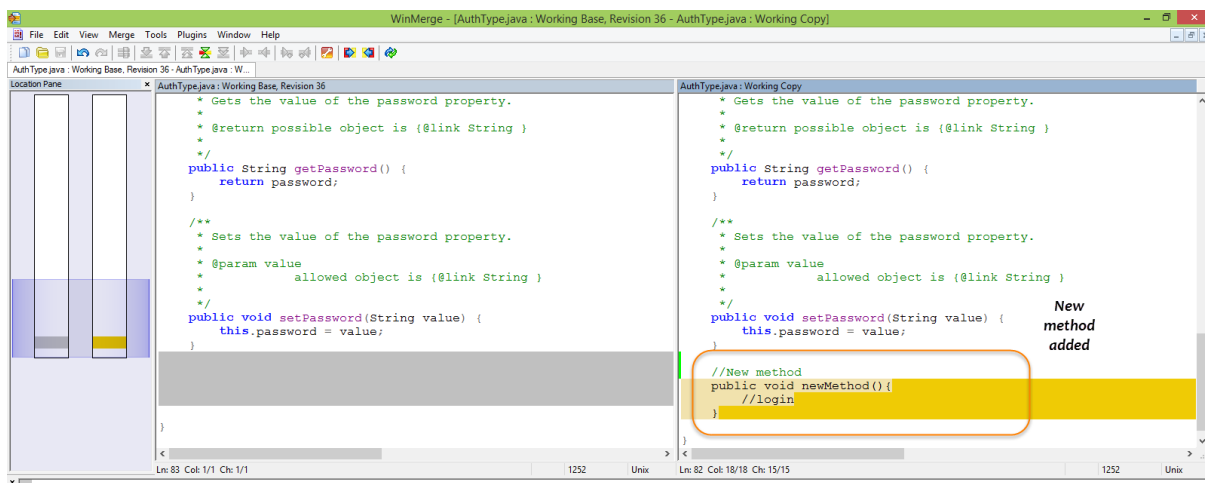
### Diff (Checking differences b/w local and repository file)

If you want to see the difference between your local changes and repo code, then use diff action.

Before commit, you can double click file and get the same

#### **Process:**

- Your file should have some local modifications
- Right Click file -> Tortoise SVN – Diff
- A comparison window would occur using which you can check the difference between the two.



### Deleting a File (Deleting file from Repository)

For deleting a file from Repo, you should use “SVN Delete” operation.

Also your delete actions should be committed otherwise the file would just be deleted in local machine but not in repo.

#### **App 1:**

- If you use keyboard “del” or “shift + del” option then the files gets deleted from local.
- With respect to SVN this file would be in missing state.
- If you right click on the folder where the file was available, you can see that deleted file with missing state.
- Right click on missing file, SVN revert, this would get the file back to you.
- But remember if you have any local changes to this file they would be lost.
- If you really want to delete the file, then you have to commit this deleted status file as well.

**App 2:**

- Right click file to be deleted -> TortoiseSVN -> delete
- In this case the file status would not be missing but rather would be deleted.
- When you click commit in this folder, you would get a popup where you see your deleted file.
- You can get it back using "Revert" action. But here as well your local changes if you have to the file would be lost.
- If you really want to delete the file, then you have to commit this deleted status file as well.

**Show Logs (Checking File History)**

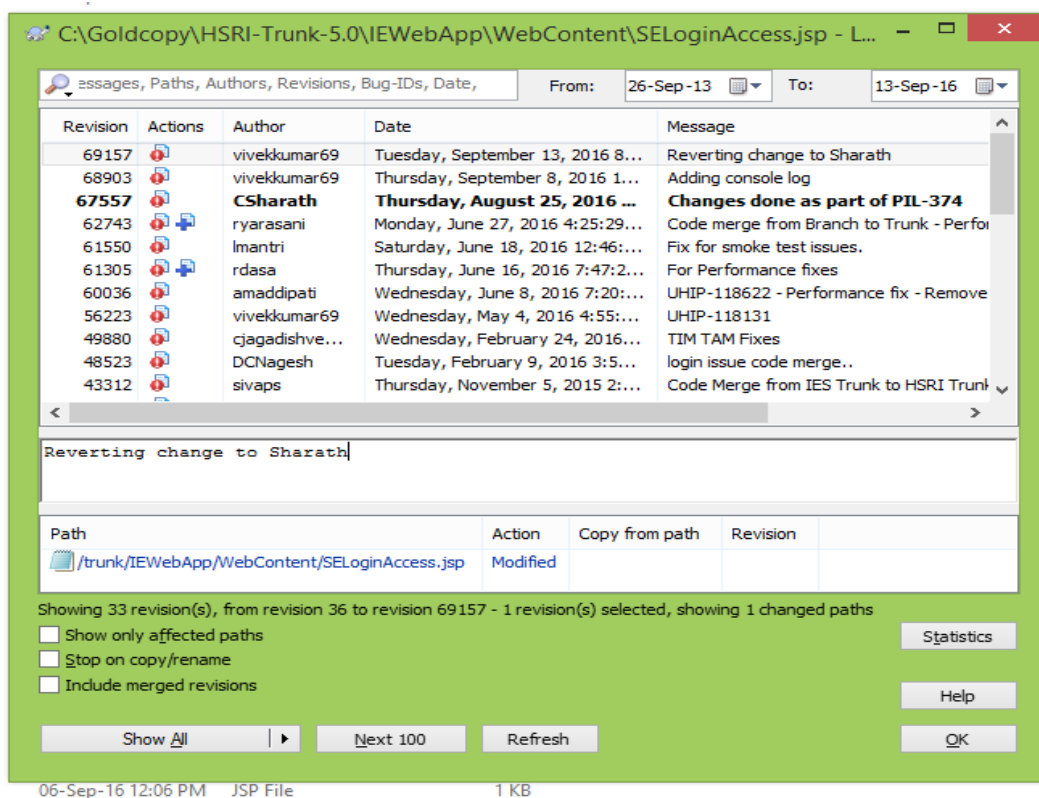
To track all changes done on repository, you can use this option.

It tells you which developer committed what changes and when.

All small changes that would be committed would be tracked and visible using this option.

**Process:**

- Right folder/file -> TortoiseSVN -> Show Logs

**Lock (Getting Control on the file)**

If very frequent commits are happening on a file, then whenever you try to commit, you would be asked to update and merge your changes w.r.t the latest repo code.

This might kill your time. What you can actually do is get a "lock" over the file you are modifying.

Get a lock would not allow anybody else to commit any changes to that file, unless that lock is released.

When somebody tries to commit to this file, they would get a message saying commit is blocked. If this person want to know who blocked it, then he should also try to get a lock on the same file. This would give a message confirming the locked person Id.

Make sure that when you lock a file, try to release the lock ASAP

**Process:**

- Right click file that you want to lock → Get Lock
- Add some lock message
- Keep working on the file
- Once work done, right click the same file and release lock.

**Revert (Bringing file to previous state)**

Reverts any local changes to a file or directory and resolves any conflicted states.

It can also change file states, like deleted ->normal or add -> un versioned

**Process:**

- If you right click and revert from file with code changes, then all changes done to the file would be lost and would be a fresh file the way to last updated
- If right clicked a missing/deleted file and revert then files would come back(with last update repo code, you changes to such files would be lost)

**Reverting File to Older Version**

Whenever you make any change to file and commit to repo, a new revision id would be created.

At any point of time if you want to go back to the old revisions then it's possible.

When you revert to older revision, first the changes would reflect in your local machine, then you have to commit those in the repo machine.

**Process:**

- Right click the file -> Show logs
- Choose the revision to which you want to revert the changes to.
- You can verify the changes by double click in the below window that gets selected on choosing the desired commit related revision.
- Right click, on the chosen rollback revision use, "revert to this revision" option.
- Locally your file would be modified to this revision.
- Now you can commit the change so its becomes available to all others who update their local code base

**Getting SVN URL of a file**

Right click on the file -> Properties -> Subversion tab give the SVN URL as to where this file is available in repository.

**Working with Branches**

In order for continuing with project work without affecting work flow of other teams you might end up working on a branch.

This branch is actually cut from the main trunk only.

And we need to understand that code from trunk only would be deployed in production.

Thus whatever coding commits happen on branch should eventually be merged to trunk.

Operation would be cutting a branch from trunk and merging branch back to trunk.

### **Cutting Branch from Trunk**

Go to the folder where you have checkout the trunk code (means folder pointing to <proj\_name>/trunk/)

Right click empty folder space there -> TortoiseSVN -> Branch/tag option

On popup, the from URL would be default set to trunk URL but you got to change the to URL

Change the "To URL": /branches/<branch name>

Note: don't create a folder with branch name under branches folder. You just type branch name in the "to URL".

Tool would automatically create a branch from the revision

### **Subclipse plugin**

As of now we have seen how to use Tortoise SVN as SVN client for code sharing. However, it would make sense to work with Subversion repositories right from your IDE.

Eclipse IDE has built-in integration with Concurrent Versions System (CVS), but not Subversion (SVN). Here's a guide to show you how to make Eclipse IDE support **Subversion (SVN)** via **Subclipse plugin**.

### **Pre-requisites to work with Subclipse**

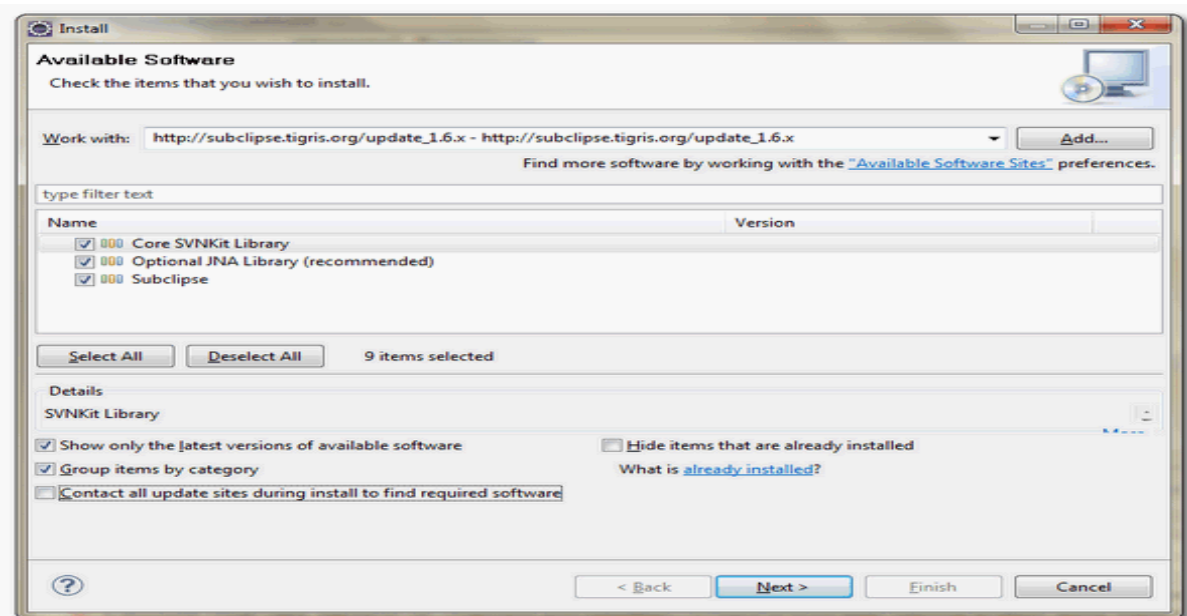
Download and install Eclipse IDE

### **Install Subclipse**

In Eclipse IDE, top menu → select "Help" → "Install New Software..." and

put [http://subclipse.tigris.org/update\\_1.6.x](http://subclipse.tigris.org/update_1.6.x) in the "work with" textbox and click on the "Add" button.

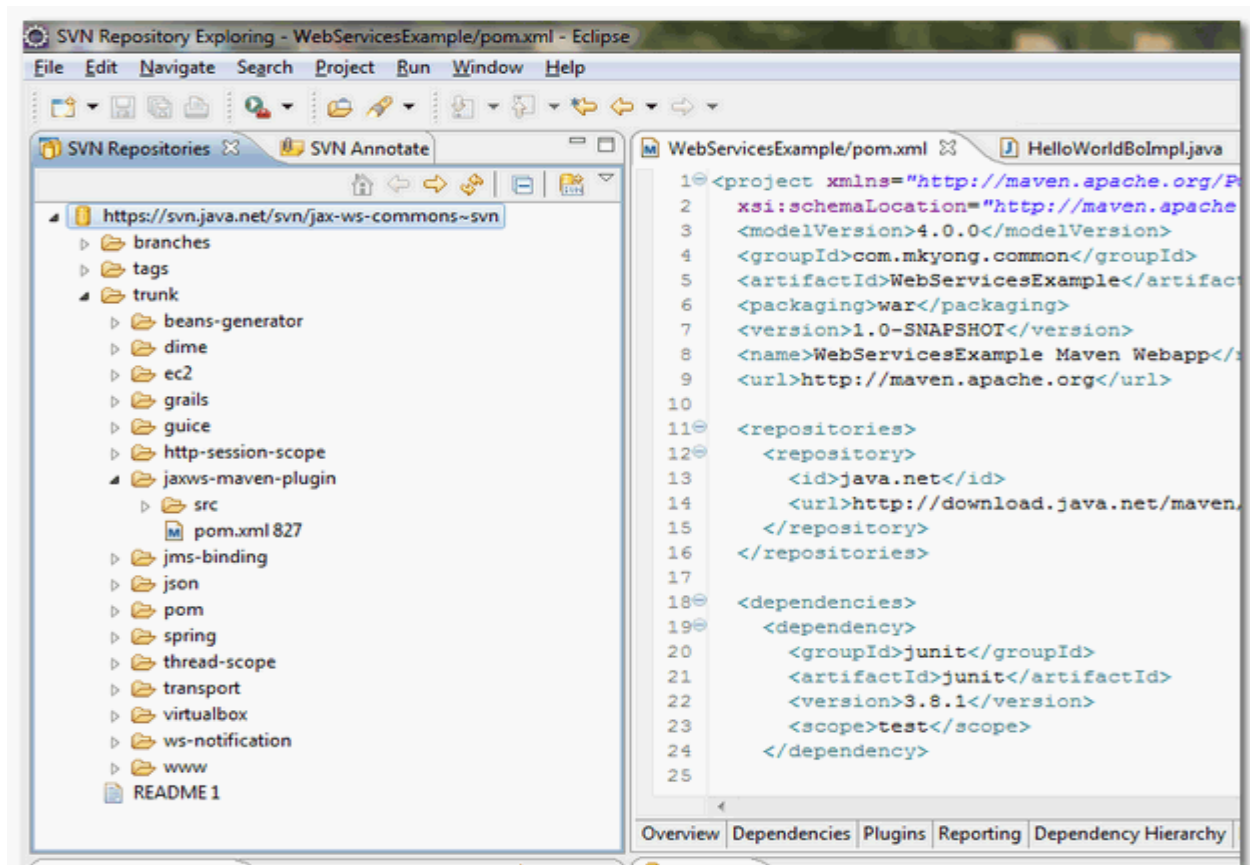
Select all components and install it.



**Once Installation is completed Re-start your Eclipse IDE to reflect latest changes**

## Test SVN in Eclipse

Eclipse IDE, top menu, select “Windows” → “Open Perspective” → “Other...”, choose “SVN Repositories”



Now, you can perform SVN functionalities in this “SVN Repositories” perspective like Check-in, Check-out, Check history and reverting to previous version etc.

==== 000 ====