1- Creating repository in the server (Version control system server)

2- Create the project folder in your local machine

3- Checkout the repository that you have created.

-Whenever you want to modify code for a project on server, you first need to create a working copy of the project files on your personal computer, and then you can modify the files in your working copy and later send those changes to the repository on the server.

-You should now have a green check mark icon next to your working directory.

-Now that you have checked out a copy of the /trunk, /tags, branches directories from the repository into your working copy directory, it is time to add some files to the repository.

4- Create two java files in your local working copy—diretory.

5- Right click on the name of the working directory and choose TortoiseSVN --> Add. In the dialog, check the box for each file you wish to add to the repository.

Click the OK button. The icons representing the files you have just added are now be marked with a blue '+' sign, and the widget1\_working directory now is marked by a red exclamation point icon, The red exclamation point indicates that the directory contains files that have been changed since the last time the working copy was updated.

At this point, you have told TortoiseSVN that you wish to add these files to the repository, but you have not yet actually added them to the repository. To do that, you have to commit the changes back to the repository.

6- After you have added or removed files or made changes to your code (and tested those changes to make sure they work), you should commit those changes back to the repository so that others can access the changes.

-In this example, you have now informed TortoiseSVN that you wish to first.java to the repository, but you now must commit that change to the repository.

-From Windows Explorer, right click on the working directory and choose SVN Commit.

-In the dialog, type in a commit message (log messages are always required). Make sure that the "Changes made" list has a check next to both of the files you want to add.

-Click the OK button. If the commit worked, you will now see a dialog that says "Finished" and that indicates that the files were added to the repository.

-If, after updating your working copy, you receive any error messages or warning messages in the dialog, there may be a conflict between a file in your working copy and the version of the file in the repository (often by someone else).

7- users can make changes to the code on their own computers, create a patch, and then attach that patch to an issue. Other users can review the changes made in the patch and determine whether they are acceptable and whether the changed code is functional.

When the various developers agree that a patch is a good addition to the project, the project maintainer can then apply the patch to his working copy of the project and then commit those changes to the repository.

8- Revision Log Dialog

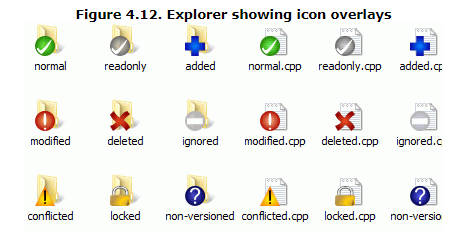
For every change you make and commit, you should provide a log message for that change. That way you can later find out what changes you made and why, and you have a detailed log for your development process.

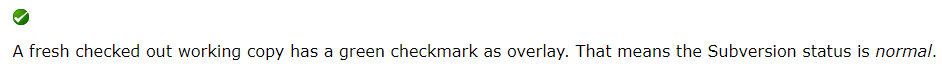
The Revision Log Dialog retrieves all those log messages and shows them to you. The display is divided into 3 panes.

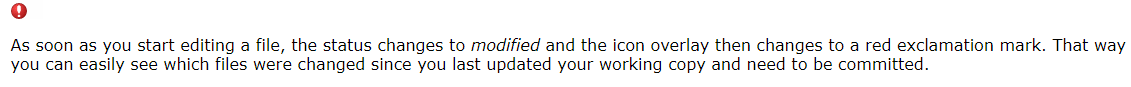
* The top pane shows a list of revisions where changes to the file/folder have been committed. This summary includes the date and time, the person who committed the revision and the start of the log message.
* Lines shown in blue indicate that something has been copied to this development line (perhaps from a branch).
* The middle pane shows the full log message for the selected revision.
* The bottom pane shows a list of all files and folders that were changed as part of the selected revision.

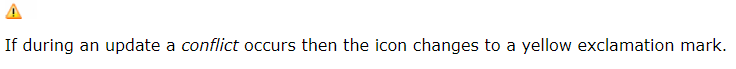
9- **Rep-Browser** Sometimes you need to work directly on the repository, without having a working copy. So the Repository Browser allows you to view the structure and status of the repository.

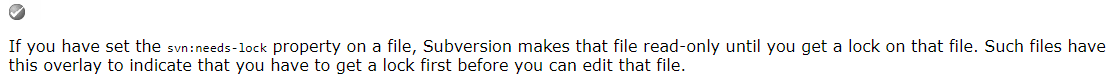
10- **Check for Modifications** While you are working on your working copy you often need to know which files you have changed/added/removed or renamed, or even which files got changed and committed by others.

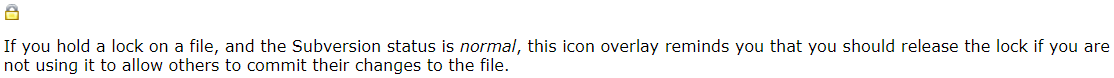


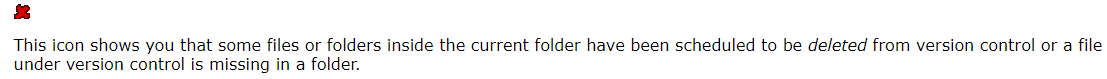


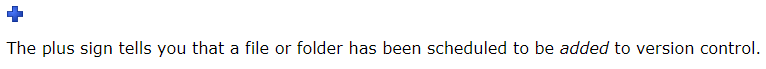


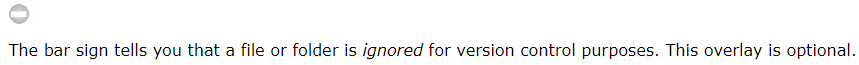


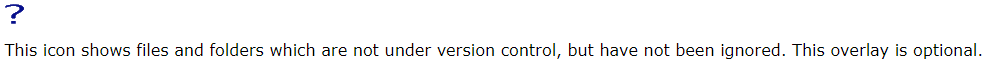








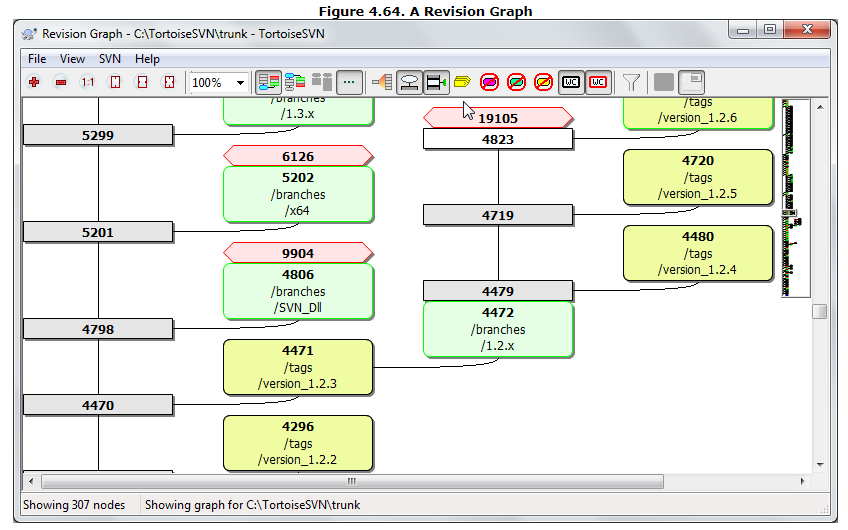




It's often very useful to know which files you have changed and also which files got changed and committed by others. That's where the command **TortoiseSVN → Check For Modifications**... comes in handy. This dialog will show you every file that has changed in any way in your working copy, as well as any unversioned files you may have.

If you click on the Check Repository then you can also look for changes in the repository. That way you can check before an update if there's a possible conflict.

11-Sometimes you need to know where branches and tags were taken from the trunk, and the ideal way to view this sort of information is as a graph or tree structure. That's when you need to use **TortoiseSVN → Revision Graph...**



12- Once in a while, you will get a conflict when you update/merge your files from the repository or when you switch your working copy to a different URL. There are two kinds of conflicts:

**file conflicts**

A file conflict occurs if two (or more) developers have changed the same few lines of a file.

**tree conflicts**

A tree conflict occurs when a developer moved/renamed/deleted a file or folder, which another developer either also has moved/renamed/deleted or just modified.

You can use the Resolved command for multiple files if you right click on the parent folder and select **TortoiseSVN → Resolved...** This will bring up a dialog listing all conflicted files in that folder, and you can select which ones to mark as resolved.

13-If after performing some action you decide that you want to undo, the undo and get your working copy back to its previous unmodified state, you should use **TortoiseSVN → Revert** from within Windows Explorer, which will discard the local modifications made by this reverse merge action.

14- **clean up** If a Subversion command cannot complete successfully, perhaps due to server problems, your working copy can be left in an inconsistent state. In that case you need to use **TortoiseSVN → Cleanup** on the folder. It is a good idea to do this at the top level of the working copy.

15- **locking** By default, nothing is locked and anyone who has commit access can commit changes to any file at any time. Others will update their working copies periodically and changes in the repository will be merged with local changes.

If you Get a Lock on a file, then only you can commit that file. Commits by all other users will be blocked until you release the lock. A locked file cannot be modified in any way in the repository, so it cannot be deleted or renamed either, except by the lock owner.