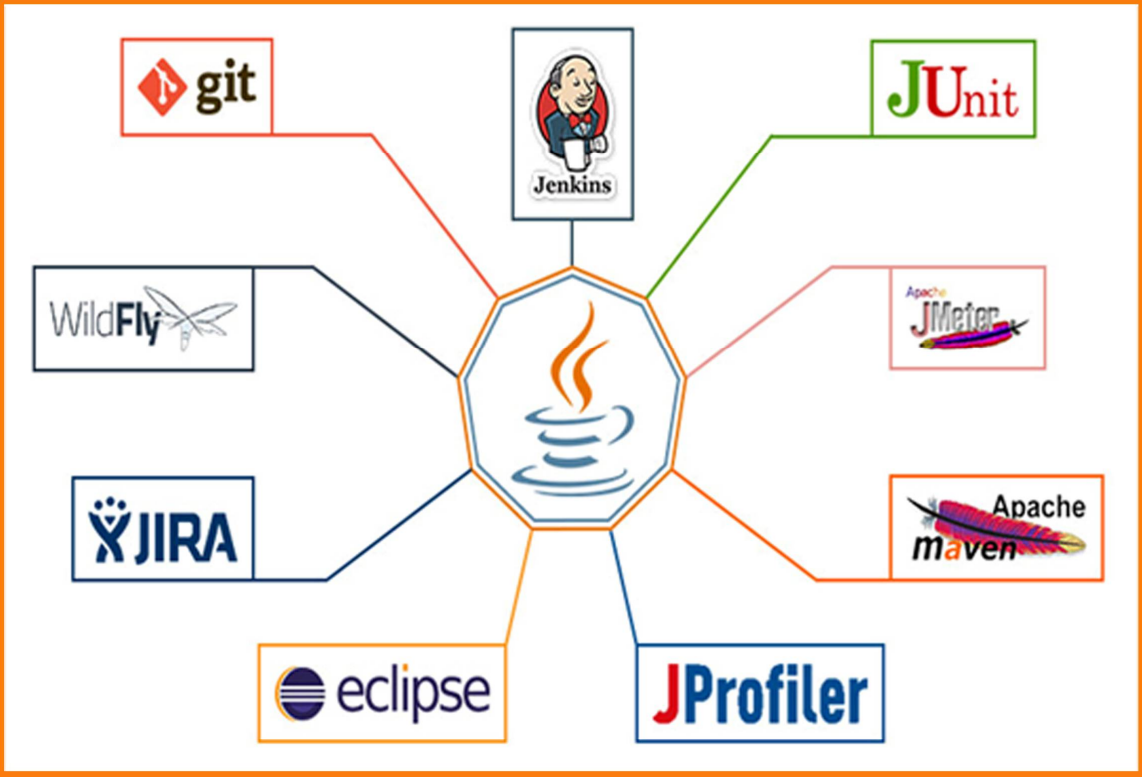
JAVA means DURGA SOFT

Java Real Time Tools

SVN



India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

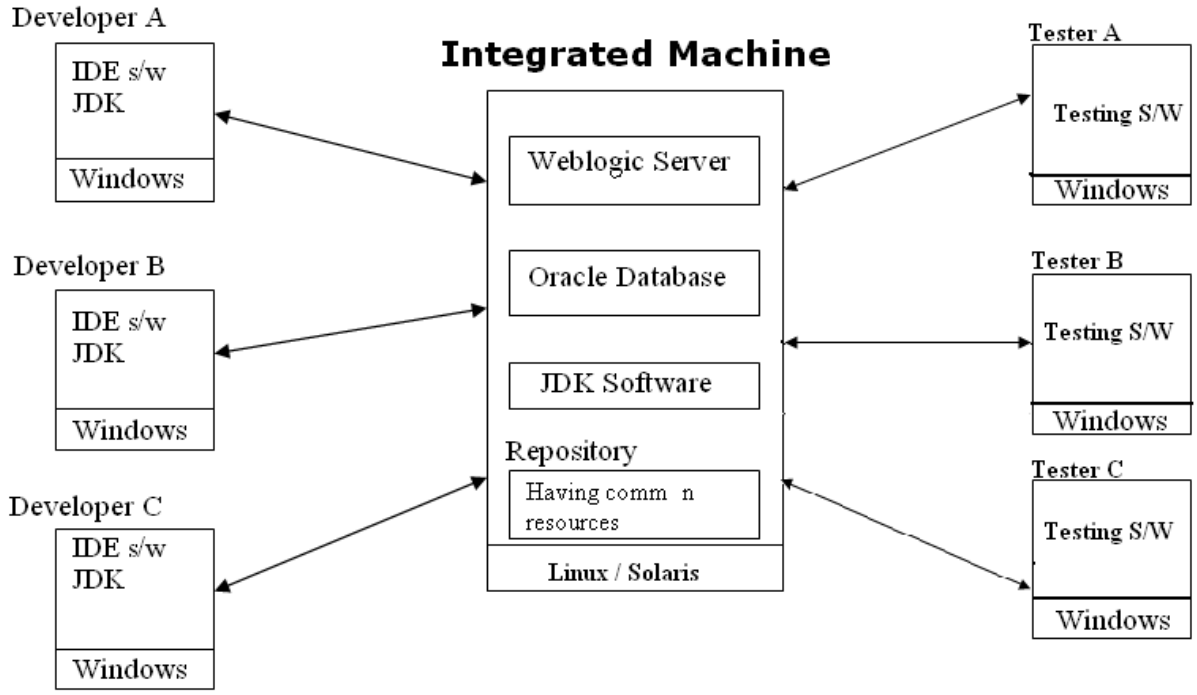
SVN

CONTENT:

SVN (Version Control System)

Introduction:

In real time all developers and testers machines will be there running in windows environment but all these machines will be connected to a common machine of company called **integrated machine.** Generally this integrated machine resides in linux or solaris environment having high configuration and also contains the common software that are required for multiple projects of company.



The multiple projects of company will use the multiple logical databases that are created in database software of integrated machine on one per project basis.

During development mode of the project, the project will be maintained in the CVS Repository or SVN Repository to make the resources of the project visible and accessible for all developers of the project.

Definition : **SVN** is a version control system. Using it, developers can record the history of their source files. This is also called as Source Code Management.

CollabNet Inc. developed SVN in 2000.

- The SVN repository keeps track of various operations that are done in the files by developers by accessing the files by developers by accessing the files from SVN repository.
- Subversion can use the HTTP-based WebDAV/DeltaV protocol for network communications, and the Apache web server to provide repository-side network service. This gives Subversion an advantage over CVS in interoperability, and provides various key features for free: authentication, path-based authorization, wire compression, and basic repository browsing.
- Depending on your Operating System, you might choose the 32-bit or 64-bit versions and download.
- SVN repository keeps track of various modifications done in files by different developers by generating versions.

Ex:

Test.java (Original file)

Test.java 1.1 (after first modification)

Test.java 1.2 (after second modification)

Test.java 1.3 (after third modification)

Goals:

There are three basic goals of a version control system (VCS):

- We want people to be able to work simultaneously, not serially.

Think of your team as a multi-threaded piece of software with each developer running in his own thread. The key to high performance in a multi-threaded system is to maximize concurrency. Our goal is to never have a thread which is blocked on some other thread.

- When people are working at the same time, we want their changes to not conflict with each other.

Multi-threaded programming requires great care on the part of the developer and special features such as critical sections, locks, and a test-and-set instruction on the CPU. Without these kinds of things, the threads would overwrite each other's data.
A multi-threaded software team needs things too, so that developers can work without messing each other up. That is what the version control system provides.

- We want to archive every version of everything that has ever existed – ever. And who did it. And when. And why.

Benefits of Source Code Management:

- ✓ All code changes are tracked.
- ✓ Avoid losing work due to simple mistakes (Allows you to roller back changes).
- ✓ Codes changes across several developers can be synchronized.
- ✓ Makes it easy to backup your source code.
- ✓ You can work on several different copies of the same application at the same time.
- ✓ Supervisor can see how the code evolved over time.

Evolution of SVN:

In early 2000, CollabNet, Inc. began seeking developers to write a replacement for CVS. CollabNet offers a collaboration software suite called CollabNet Enterprise Edition (CEE)**2** of which one component is version control. Although CEE used CVS as its initial version control system, CVS's limitations were obvious from the beginning, and CollabNet knew it would eventually have to find something better. Unfortunately, CVS had become the de facto standard in the open source world largely because there wasn't anything better, at least not under a free license. So CollabNet

determined to write a new version control system from scratch, retaining the basic ideas of CVS, but without the bugs and misfeatures.

When CollabNet called, Karl immediately agreed to work on the project, and Jim got his employer, Red Hat Software, to essentially donate him to the project for an indefinite period of time. CollabNet hired Karl and Ben Collins-Sussman, and detailed design work began in May. With the help of some well-placed prods from Brian Behlendorf and Jason Robbins of CollabNet, and Greg Stein (at the time an independent developer active in the WebDAV/DeltaV specification process), Subversion quickly attracted a community of active developers and welcomed the chance to finally do something about it.

After fourteen months of coding, Subversion became "self-hosting" on August 31, 2001. That is, Subversion developers stopped using CVS to manage Subversion's own source code, and started using Subversion instead.

Subversion's Architecture:

On one end is a Subversion repository that holds all of your versioned data. On the other end is your Subversion client program, which manages local reflections of portions of that versioned data (called "working copies"). Between these extremes are multiple routes through various Repository Access (RA) layers. Some of these routes go across computer networks and through network servers which then access the repository. Others bypass the network altogether and access the repository directly.

Subversion Installation:

Subversion is built on a portability layer called APR—the Apache Portable Runtime library. The APR library provides all the interfaces that Subversion needs to function on different operating systems: disk access, network access, memory management, and so on. While Subversion is able to use Apache as one of its network server programs, its dependence on APR does not mean that Apache is a required component. APR is a standalone library useable by any application. It does mean, however, that like Apache, Subversion clients and servers run on any operating system that the Apache httpd server runs on: Windows, Linux, all flavors of BSD, Mac OS X, Netware, and others.

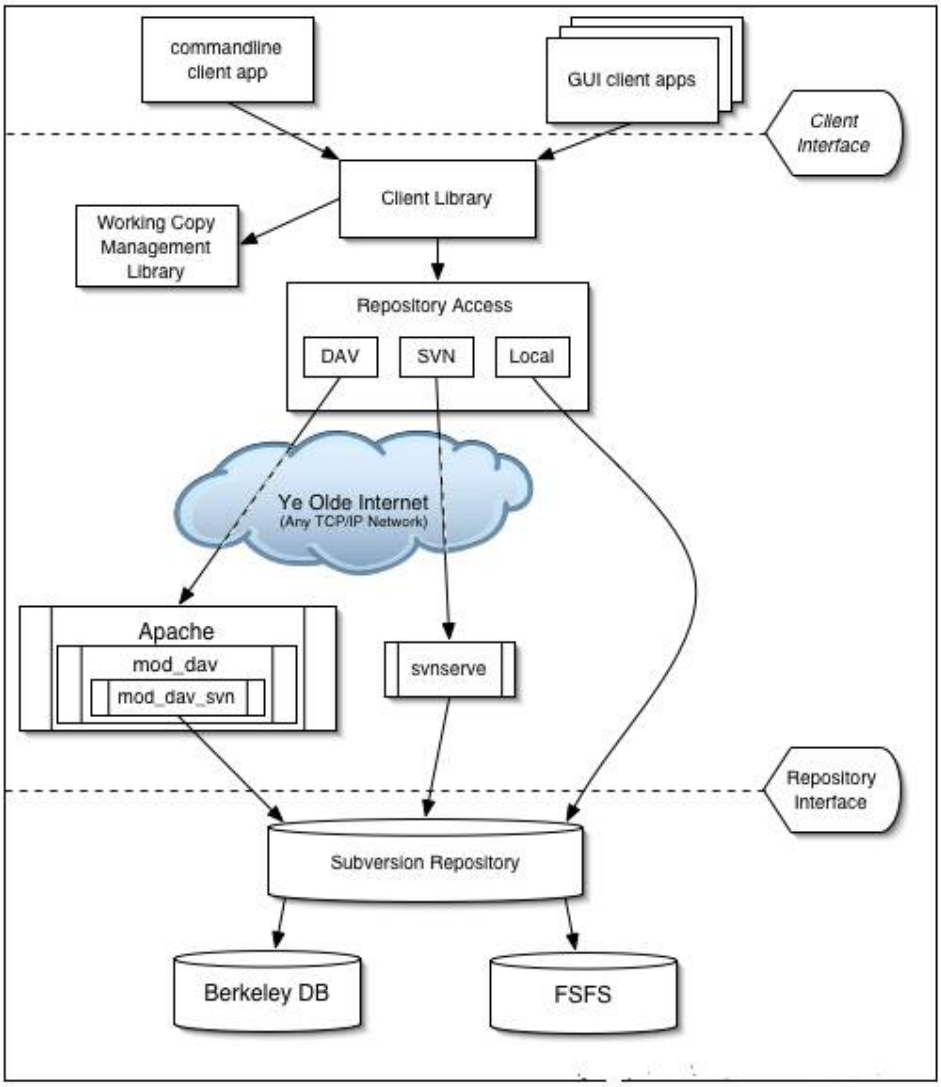
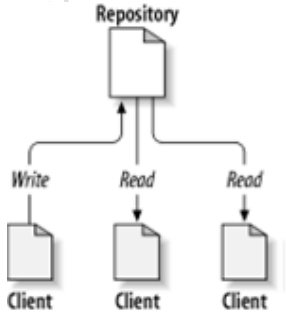


Figure 1.1 Subversion's Architecture

Repository:

Subversion is a centralized system for sharing information. At its core is a repository, which is a central store of data. The repository stores information in the form of a filesystem tree—a typical hierarchy of files and directories. Any number of clients connect to the repository, and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others. Figure illustrates this.



What makes the Subversion repository special is that it remembers every change ever written to it: every change to every file, and even changes to the directory tree itself, such as the addition, deletion, and rearrangement of files and directories.
When a client reads data from the repository, it normally sees only the latest version of the filesystem tree. But the client also has the ability to view previous states of the filesystem.
For example, a client can ask historical questions like, "What did this directory contain last Wednesday?" or "Who was the last person to change this file, and what changes did he make?".
These are the sorts of questions that are at the heart of any version control system: systems that are designed to record and track changes to data over time.

Versioning Models:

The core mission of a version control system is to enable collaborative editing and sharing of data. But different systems use different strategies to achieve this.

Problem of File-Sharing:

All version control systems have to solve the same fundamental problem: how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet? It's all too easy for users to accidentally overwrite each other's changes in the repository. Consider the scenario, Suppose we have two co-workers, Harry and Sally.

They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, then it's possible that (a few moments later) Sally could accidentally overwrite them with her own new version of the file. While Harry's version of the file won't be lost forever (because the system remembers every change), any changes Harry made won't be present in Sally's newer version of the file, because she never saw Harry's changes to begin with. Harry's work is still effectively lost—or at least missing from the latest version of the file—and probably by accident. This is definitely a situation we want to avoid!

Lock-Modify-Unlock Solution:

Many version control systems use a lock-modify-unlock model to address the problem of many authors clobbering each other's work. In this model, the repository allows only one person to change a file at a time. This exclusivity policy is managed using locks. Harry must "lock" a file before he can begin making changes to it. If Harry has locked a file, then Sally cannot also lock it, and therefore cannot make any changes to that file. All she can do is read the file, and wait for Harry to finish his changes and release his lock. After Harry unlocks the file, Sally can take her turn by locking and editing the file. Figure demonstrates this simple solution.

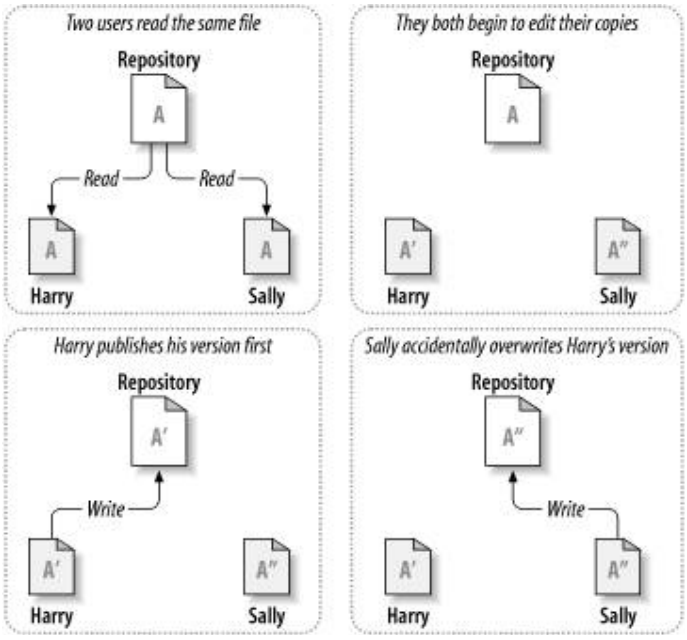


Figure 2.2 The problem to avoid

The problem with the lock-modify-unlock model is that it's a bit restrictive, and often becomes a roadblock for users:

• **Locking may cause administrative problems.**

Sometimes Harry will lock a file and then forget about it. Meanwhile, because Sally is still waiting to edit the file, her hands are tied. And then Harry goes on vacation. Now Sally has to get an administrator to release Harry's lock. The situation ends up causing a lot of unnecessary delay and wasted time.

• **Locking may cause unnecessary serialization.**

What if Harry is editing the beginning of a text file, and Sally simply wants to edit the end of the same file? These changes don't overlap at all. They could easily edit the file simultaneously, and no great harm would come, assuming the changes were properly merged together. There's no need for them to take turns in this situation.

• **Locking may create a false sense of security.**

Pretend that Harry locks and edits file A, while Sally simultaneously locks and edits file B. But suppose that A and B depend on one another, and the changes made to each are semantically incompatible. Suddenly A and B don't work together anymore. The locking system was powerless to prevent the problem—yet it somehow provided a false sense of security. It's easy for Harry and Sally to imagine that by locking files, each is beginning a safe, insulated task, and thus not bother discussing their incompatible changes early on.

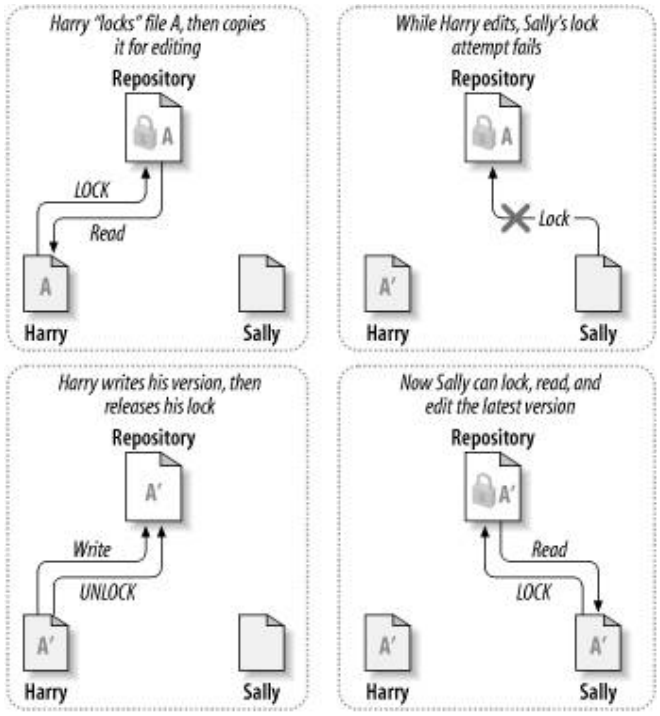


Figure 2.3 lock-modify-unlock solution

Copy-Modify-Merge Solution:

Subversion, and other version control systems use a copy-modify-merge model as an alternative to locking. In this model, each user's client contacts the project repository and creates a personal working copy—a local reflection of the repository's files and directories.

Users then work in parallel, modifying their private copies. Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a human being is responsible for making it happen correctly.
Here's an example. Say that Harry and Sally each create working copies of the same project, copied from the repository. They work concurrently, and make changes to the same file A within their copies. Sally saves her changes to the repository first. When Harry attempts to save his changes later, the repository informs him that his file A is out-of-date. In other words, that file A in the repository has somehow changed since he last copied it. So Harry asks his client to merge any new changes from the repository into his working copy of file A. Chances are that Sally's changes don't overlap with his own; so once he has both sets of changes integrated, he saves his working copy back to the repository. **Figure 2.4** and **Figure 2.5** show this process.

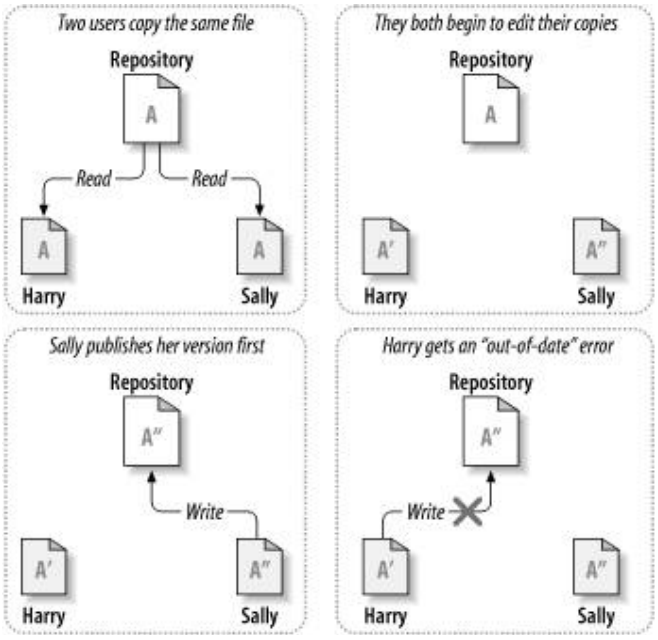


Figure 2.4 copy-modify-merge solution

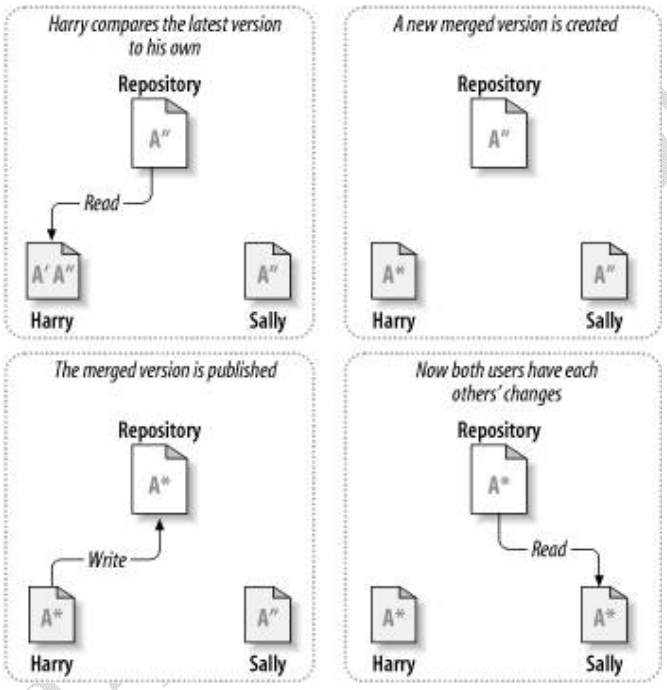


Figure 2.5 copy-modify-merge solution.

But what if Sally's changes do overlap with Harry's changes? What then? This situation is called a conflict, and it's usually not much of a problem. When Harry asks his client to merge the latest repository changes into his working copy, his copy of file A is somehow flagged as being in a state of conflict: he'll be able to see both sets of conflicting changes, and manually choose between them. Note that software can't automatically resolve conflicts; only humans are capable of understanding and making the necessary intelligent choices. Once Harry has manually resolved the overlapping changes—perhaps after a discussion with Sally—he can safely save the merged file back to the repository.

The copy-modify-merge model may sound a bit chaotic, but in practice, it runs extremely smoothly. Users can work in parallel, never waiting for one another. When they work on the same files, it turns out that most of their concurrent changes don't overlap at all; conflicts are infrequent. And the amount of time it takes to resolve conflicts is far less than the time lost by a locking system.

Subversion's Components:

Svn

The command-line client program

svnversion

A program for reporting the state (in terms of revisions of the items present) of a working copy

svnlook

A tool for directly inspecting a Subversion repository

svnadmin

A tool for creating, tweaking, or repairing a Subversion repository

mod_dav_svn

A plug-in module for the Apache HTTP Server, used to make your repository available to others over a network

svnserve

A custom standalone server program, runnable as a daemon process or invokable by SSH; another way to make your repository available to others over a network

svndumpfilter

A program for filtering Subversion repository dump streams

svnsync

A program for incrementally mirroring one repository to another over a network

svnrdump

A program for performing repository history dumps and loads over a network

Basic Work Cycle:

The most common things that you might find yourself doing with Subversion in the course of a day's work.

- Update your working copy
  - svn update

- Make changes
  - svn add
  - svn delete
  - svn copy
  - svn move

- Examine your changes
  - svn status
  - svn diff
  - svn revert

- Merge others' changes
  - svn merge
  - svn resolved

- Commit your changes
  - svn commit

Access Subversion Repositories:

Subversion repositories can be accessed through many different methods—on local disk, or through various network protocols.

Schema	Access Method
file:///	direct repository access (on local disk).
http://	access via WebDAV protocol to Subversion-aware Apache server
Svn://	access via custom protocol to an svnserve server
https://	same as http://, but with SSL encryption.
Svn+ssh://	same as svn://, but through an SSH tunnel

Working Copies Track the Repository:

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, and if they're source code files, you can compile your program from them in the usual way. Your working copy is your own private work area: Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so. You can even have multiple working copies of the same project.

After you've made some changes to the files in your working copy and verified that they work properly, Subversion provides you with commands to "publish" your changes to the other people working with you on your project (by writing to the repository). If other people publish their own changes, Subversion provides you with commands to merge those changes into your working directory (by reading from the repository).

Given this information, by talking to the repository, Subversion can tell which of the following four states a working file is in:

Unchanged, and current

The file is unchanged in the working directory, and no changes to that file have been committed to the repository since its working revision. An **svn commit** of the file will do nothing, and an **svn update** of the file will do nothing.

Locally changed, and current

The file has been changed in the working directory, and no changes to that file have been committed to the repository since its base revision. There are local changes that have not been committed to the repository, thus an **svn commit** of the file will succeed in publishing your changes, and an **svn update** of the file will do nothing.

Unchanged, and out-of-date

The file has not been changed in the working directory, but it has been changed in the repository. The file should eventually be updated, to make it current with the public revision. An **svn commit** of the file will do nothing, and an **svn update** of the file will fold the latest changes into your working copy.

Locally changed, and out-of-date

The file has been changed both in the working directory, and in the repository. An **svn commit** of the file will fail with an "out-of-date" error. The file should be updated first; an **svn update** command will attempt to merge the public changes with the local changes. If Subversion can't complete the merge in a plausible way automatically, it leaves it to the user to resolve the conflict.

svn status command will show you the state of any item in your working copy.

Revision Keywords:

The Subversion client understands a number of revision keywords. These keywords can be used instead of integer arguments to the --revision switch, and are resolved into specific revision numbers by Subversion:

HEAD
The latest (or "youngest") revision in the repository.

BASE
The revision number of an item in a working copy. If the item has been locally modified, the "BASE version" refers to the way the item appears without those local modifications.

COMMITTED
The most recent revision prior to, or equal to, BASE, in which an item changed.

PREV
The revision immediately before the last revision in which an item changed. (Technically, COMMITTED-1).

Note: *PREV, BASE, and COMMITTED can be used to refer to local paths, but not to URLs.*

Examining History:

There are several commands that can provide you with historical data from the repository:

svn log

Shows you broad information: log messages with date and author information attached

to revisions, and which paths changed in each revision.

svn diff

Shows you the specific details of how a file changed over time.
The three distinct uses of **svn diff**:
• Examine local changes
• Compare your working copy to the repository
• Compare repository to repository

svn cat

To examine an earlier version of a file and not necessarily the differences between two files,

svn list

Displays the files in a directory for any given revision.

Branching and Merging

Branching, tagging, and merging are concepts common to almost all version control systems. Branching is a fundamental part of version control.

Branch:

The basic concept of branch—namely, a line of development that exists independently of another line, yet still shares a common history if you look far enough back in time. A branch always begins life as a copy of something, and moves on from there, generating its own history

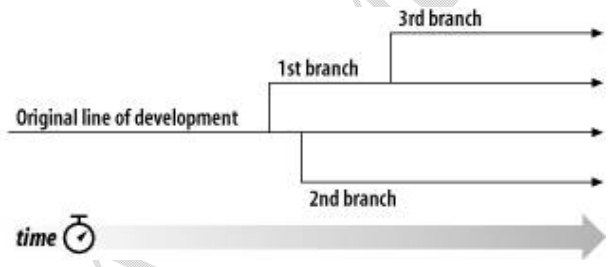


Figure Branches of development

Subversion has commands to help you maintain parallel branches of your files and directories. It allows you to create branches by copying your data, and remembers that the copies are related to one another. It also helps you duplicate changes from one branch to another. Finally, it can make portions of your working copy reflect different branches, so that you can “mix and match” different lines of development in your daily work.

Creating a Branch:

Creating a branch is very simple: you make a copy of the project in the repository using the **svn copy** command. Subversion is not only able to copy single files, but whole directories as well.

There are two different ways to make a copy. We'll demonstrate the messy way first, just to make the concept clear.
To begin, check out a working copy of the project's root directory, /calc:

```
$ svn checkout http://svn.example.com/repos/calc bigwc
A bigwc/trunk/
A bigwc/trunk/Makefile
A bigwc/trunk/integer.c
A bigwc/trunk/button.c
A bigwc/branches/
Checked out revision 340.
```

Making a copy is now simply a matter of passing two working-copy paths to the **svn copy** command:

svn copy is able to operate directly on two URLs.

```
$ svn copy http://svn.example.com/repos/calc/trunk \
http://svn.example.com/repos/calc/branches/my-calc-branch \
$ svn status
A + branches/my-calc-branch
Committed revision 341.
```

Cheap Copies:

Subversion's repository has a special design. When you copy a directory, you don't need to worry about the repository growing huge—Subversion doesn't actually duplicate any data. Instead, it creates a newdirectory entry that points to an existing tree. If you're a Unix user, this is the same concept as a hard-link. From there, the copy is said to be “lazy”. That is, if you commit a change to one file within the copied directory, then only that file changes—the rest of the files continue to exist as links to the original files in the original directory.
It takes a very tiny, constant amount of time to make a copy of it. In fact, this feature is the basis of how commits work in Subversion: each revision is a “cheap copy” of the previous revision, with a few items lazily changed within.

Creating the Layout, and Importing Initial Data:

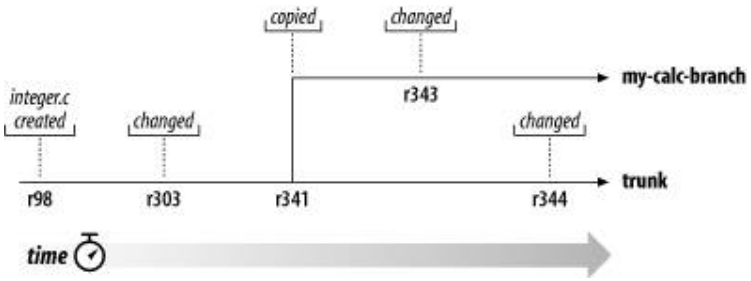
After deciding how to arrange the projects in your repository, you'll probably want to actually populate the repository with that layout and with initial project data.

.svn directory:

Every directory in a working copy contains an administrative area, a subdirectory named .svn Usually, directory listing commands won't show this subdirectory, but it is nevertheless an important directory. Whatever you do, don't delete or change anything in the administrative area! Subversion depends on it to manage your working copy.

The branching of one file's history:

Things get interesting when you look at the history of changes made to your copy of integer.c:



Copying Changes Between Branches:

For projects that have a large number of contributors, it's common for most people to have working copies of the trunk. Whenever someone needs to make a long-running change that is likely to disrupt the trunk, a standard procedure is to create a private branch and commit changes there until all the work is complete.

Key Concepts Behind Branches:

There are two important lessons that you should remember.

1. Unlike many other version control systems, Subversion's branches exist as normal filesystem directories in the repository, not in an extra dimension. These directories just happen to carry some extra historical information.

2. Subversion has no internal concept of a branch—only copies. When you copy a directory, the resulting directory is only a "branch" because you attach that meaning to it. You may think of the directory differently, or treat it differently, but to Subversion it's just an ordinary directory that happens to have been created by copying.

Merging:

The term "**merge**" somehow denotes that branches are combined together.

The **svn merge** command is almost exactly the same. Instead of printing the differences to your terminal, however, it applies them directly to your working copy as local modifications.

Svn diff-and-apply, Compares two repository trees and differences are applied to a working copy. The command takes three arguments:

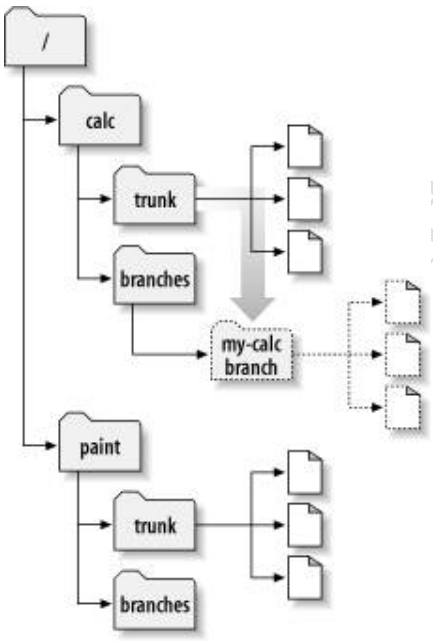
1. An initial repository tree (often called the left side of the comparison),
2. A final repository tree (often called the right side of the comparison),
3. A working copy to accept the differences as local changes (often called the target of the merge).

Once these three arguments are specified, the two trees are compared, and the resulting differences are applied to the target working copy as local modifications. When the command is done.

The syntax of **svn merge** allows you to specify the three necessary arguments rather flexibly. Here are some examples:

```
$ svn merge http://svn.example.com/repos/branch1@150 \
http://svn.example.com/repos/branch2@212 \     my-working-copy
$ svn merge -r 100:200 http://svn.example.com/repos/trunk my-working-copy
$ svn merge -r 100:200 http://svn.example.com/repos/trunk
```

The first syntax lays out all three arguments explicitly, naming each tree in the form URL@REV and naming the working copy target. The second syntax can be used as a shorthand for situations when you're comparing two different revisions of the same URL. The last syntax shows how the working-copy argument is optional; if omitted, it defaults to the current directory.



Common Branching Patterns:

Branch Releases:

Most software has a typical lifecycle: code, test, release, repeat. There are two problems with this process. First, developers need to keep writing new features while quality-assurance teams take time to test supposedly-stable versions of the software. New work cannot halt while the software is tested. Second, the team almost always needs to support older, released versions of software; if a bug is discovered in the latest code, it most likely exists in released versions as well, and customers will want to get that bugfix without having to wait for a major
new release.
Here's where version control can help. The typical procedure looks like this:

• Developers commit all new work to the trunk.
Day-to-day changes are committed to /trunk: new features, bugfixes, and so on.
• The trunk is copied to a "release" branch.
When the team thinks the software is ready for release (say, a 1.0 release), then /trunk might be copied to /branches/1.0.
• Teams continue to work in parallel.
One team begins rigorous testing of the release branch, while another team continues new work (say, for version 2.0) on /trunk. If bugs are discovered in either location, fixes are ported back and forth as necessary. At some point, however, even that process stops. The branch is "frozen" for final testing right before a release.
• The branch is tagged and released. When testing is complete, /branches/1.0 is copied to /tags/1.0.0 as a reference snapshot. The tag is packaged and released to customers.
• The branch is maintained over time. While work continues on /trunk for version 2.0, bugfixes continue to be ported from /trunk to /branches/1.0. When enough bugfixes have accumulated, management may decide to do a 1.0.1 release: /branches/1.0 is copied to /tags/1.0.1, and the tag is packaged and released.

This entire process repeats as the software matures: when the 2.0 work is complete, a new 2.0 release branch is created, tested, tagged, and eventually released. After some years, the repository ends up with a number of release branches in "maintenance" mode, and a number of tags representing final shipped versions.

Tags:

Another common version control concept is a tag. A tag is just a "snapshot" of a project in time. In Subversion, this idea already seems to be everywhere. Each repository revision is exactly that—a snapshot of the filesystem after each commit.

However, people often want to give more human-friendly names to tags, like release-1.0. And they want to make snapshots of smaller subdirectories of the filesystem. After all, it's not so easy to remember that release-1.0 of a piece of software is a particular subdirectory of revision 4822.

Syntax:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
http://svn.example.com/repos/calc/tags/release-1.0 \
-m "Tagging the 1.0 release of the 'calc' project."
Committed revision 351.
```

In Subversion, there's no difference between a tag and a branch. Both are just ordinary directories that are created by copying. Just as with branches, the only reason a copied directory is a "tag" is because humans have decided to treat it that way: as long as nobody ever commits to the directory, it forever remains a snapshot. If people start committing to it, it becomes a branch.

Branch Maintenance:

Subversion is extremely flexible. Because it implements branches and tags with the same underlying mechanism (directory copies), and because branches and tags appear in normal filesystem space, many people find Subversion intimidating. It's almost *too* flexible. In this section, we'll offer some suggestions for arranging and managing your data over time.

Repository Layout:

There are some standard, recommended ways to organize a repository. Most people create a trunk directory to hold the "main line" of development, a branches directory to contain branch copies, and a tags directory to contain tag copies. If a repository holds only one project, then often people create these top-level directories :

/trunk
/branches
/tags

If a repository contains multiple projects, admins typically index their layout by project (see **Section 5.4.1** to read more about "**project roots**"):

/paint/trunk
/paint/branches
/paint/tags
/calc/trunk
/calc/branches
/calc/tags

Terminology:

- **Repository** : area on the server where the files are stored
- **Sandbox** : a local copy of the code which you work on and then commit to the repository
- **Checkout** : Process of collecting resource or project from SVN repository.
- **Commit** : Process of keeping resources back into SVN Repository after doing modifications is called as commit operation or Checkin operation.
- **Update** : getting code changes that have been committed since you checked out the project
- **Merge** : combining changes between two versions of the same file
- **History** : shows a list of commit messages, times and, who committed for a particular file

- **Revision** : SVN assigned version number for a file

Software Description:

Some SVN Repository softwares are

- ✓ CVSNT
- ✓ WinCVS
- ✓ Tortoise
- ✓ ClearCase

❖ The SVN repository software will be installed on the integrated machine and the IDE software of the developer machines will be configured to interact with SNV Repository.

TORTOISE SVN :

Type : repository software
Version : 1.7.x
Download from : http://tortoisesvn.net/downloads.html

REPOSITORY OPERATIONS:

Atomic Commits : Atomic Commits means that, If an operation on the repository is interrupted in the middle, the repository will not be left in an inconsistent state. Are the check-in operations atomic? Are the check-in operations atomic, or can interrupting an operation leave the repository in an intermediate state?

CVS	Commits are not atomic
SVN	Commits ate atomic

Files and Directories Moves or Renames: Does the system support moving a file or directory to a different location while still retaining the history of the file?

CVS	No. Renames are not supported and a manual one may break history in two.
SVN	Yes. Renames are supported.

Files and Directories Copies: Does the version control system supports copying files or directories to a different location at the repository level, while retaining the history?

CVS	No. copies are not supported.
SVN	Yes. It's a very cheap operation (O(1)) that is also utilized for branching.

Remote Repository Replication: Does the system support cloning a remote repository to get a functionally equivalent copy in the local system? That should be done without any special access to the remote server except for normal repository access.

CVS	No
SVN	Indirectly, by using the SVN::Mirror script [2] or the svn-push utility [3].

Propagating Changes to Remote Repositories: Can the system propagate changes from one repository to another?

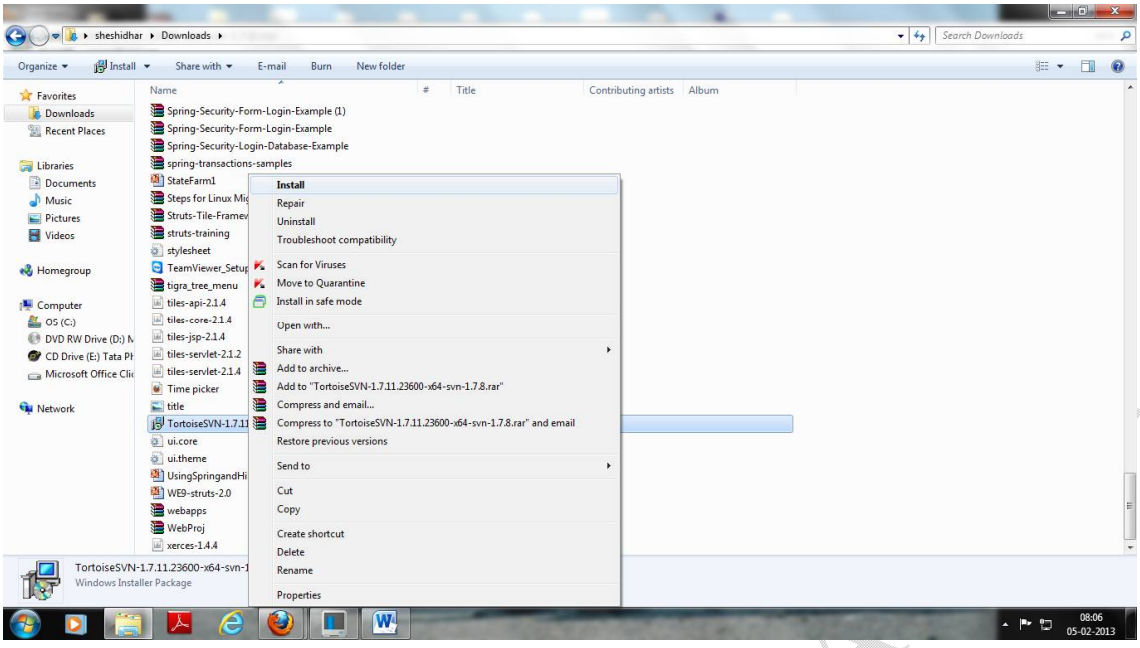
CVS	No
SVN	Yes, using either the SVN::Mirror script [2] or the svn-push utility [3].

Repository Permissions: Is it possible to define permissions on access to different parts of a remote repository? Or is access open for all?

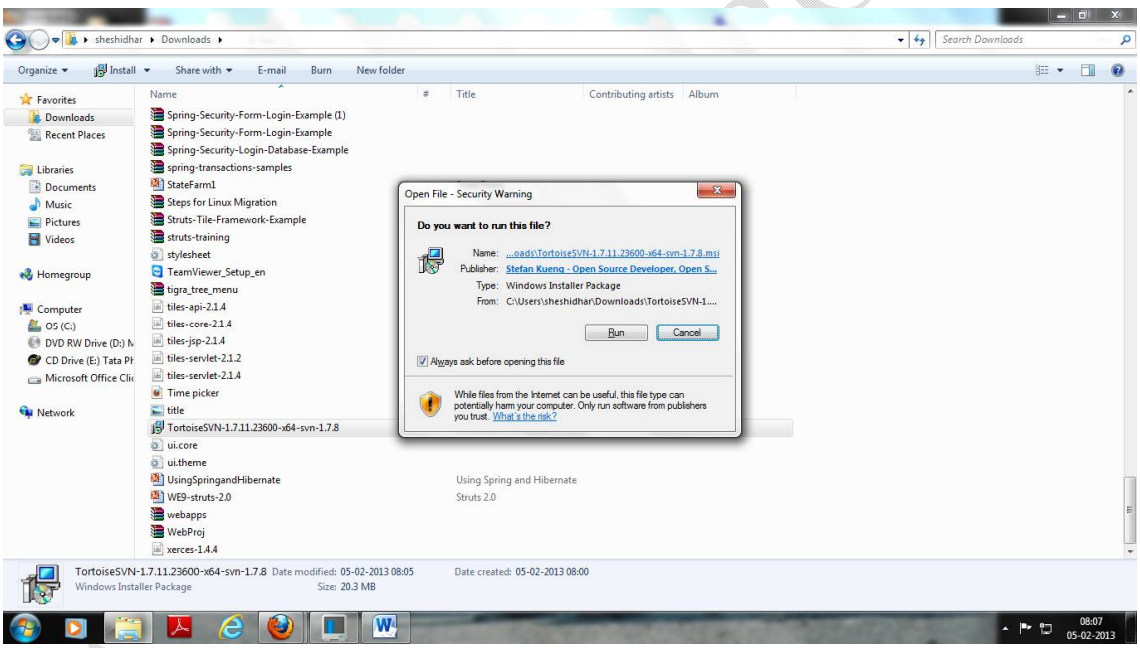
CVS	Limited. Pre-commit hook scripts' can be used to implement various permissions systems.
SVN	Yes. The WebDAV-based service supports defining HTTP permissions various directories of the repository.

TORTOISE SVN INSTALLATION:

SERVER INSTALLATION STEPS

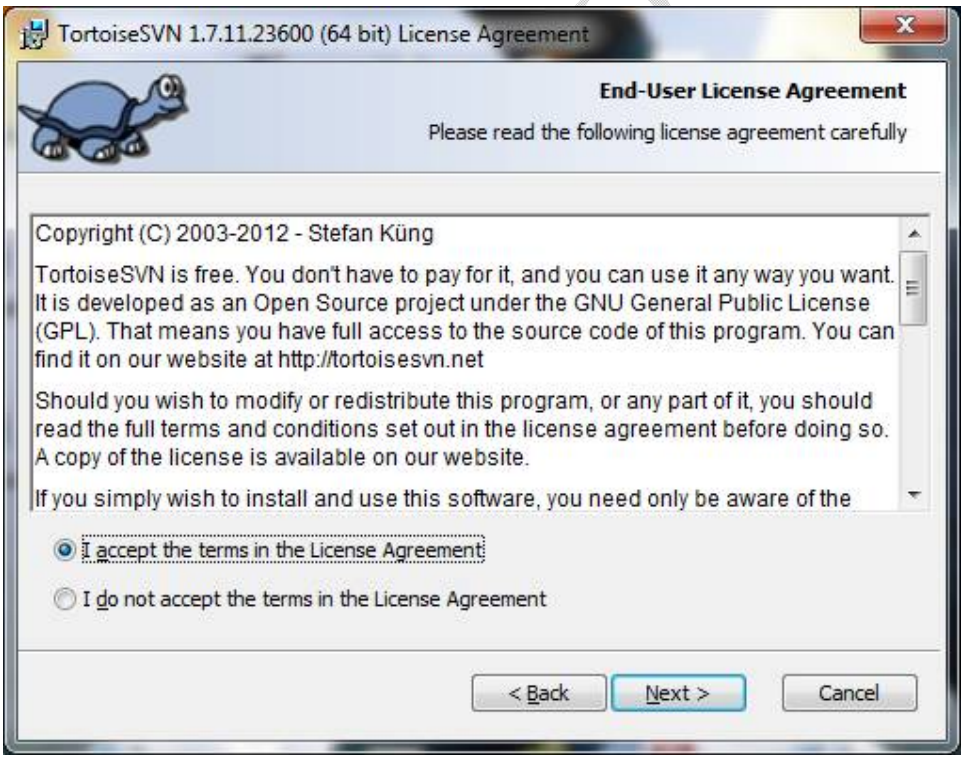


Right click on **TortoiseSVN-1.7.11.23600-x64-svn-1.7.8 c**lick on install.

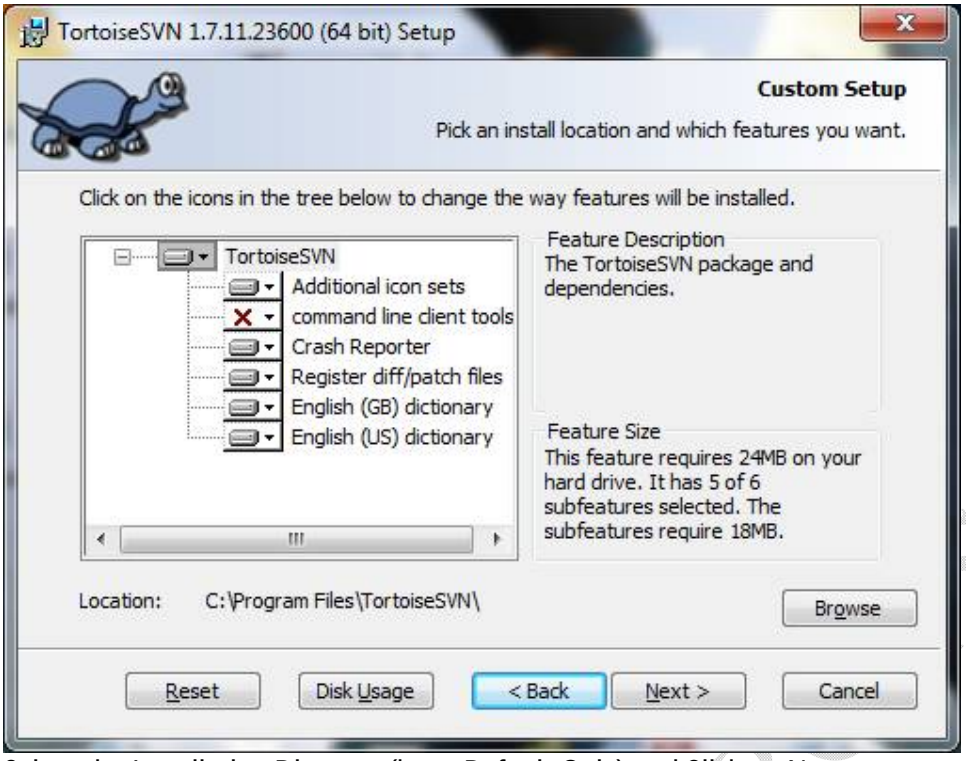


Click on Run.

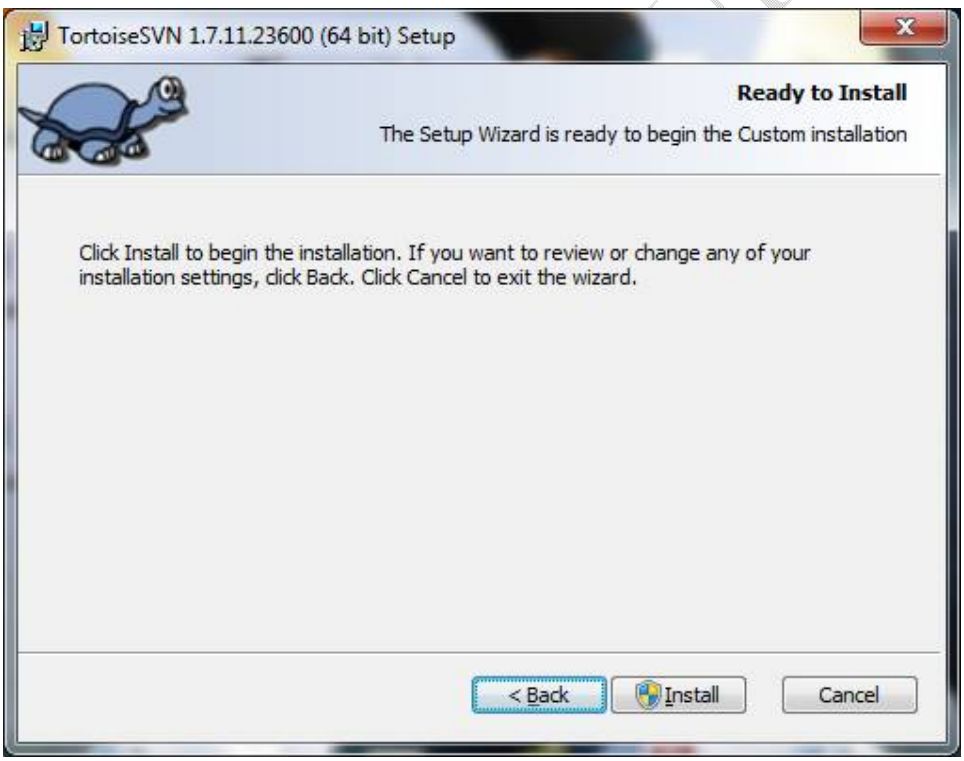
Click on Next



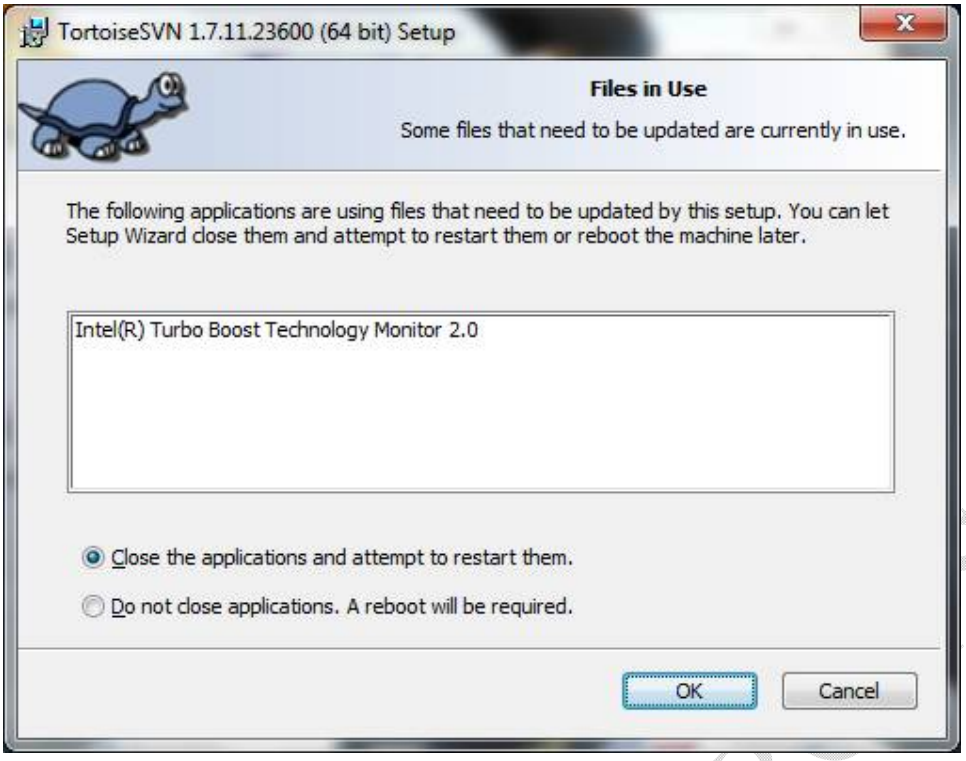
Accept the agreement and click on Next



Select the Installation Directory(keep Default Only) and Click on Next



Click On Install



Click on OK

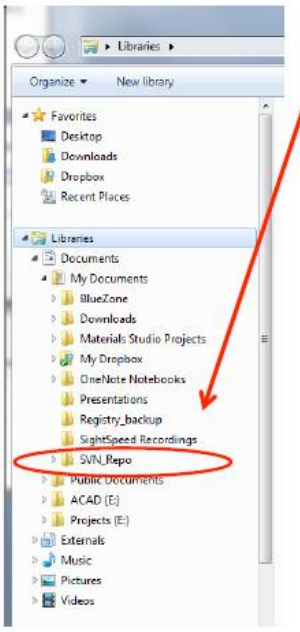
Click on Finish

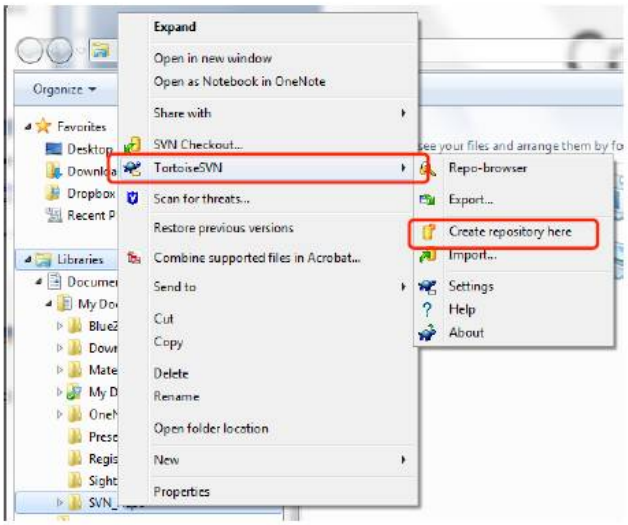
After installing **TortoiseSVN** restart the system.

Procedure to create SVN Repository for certain project / module :

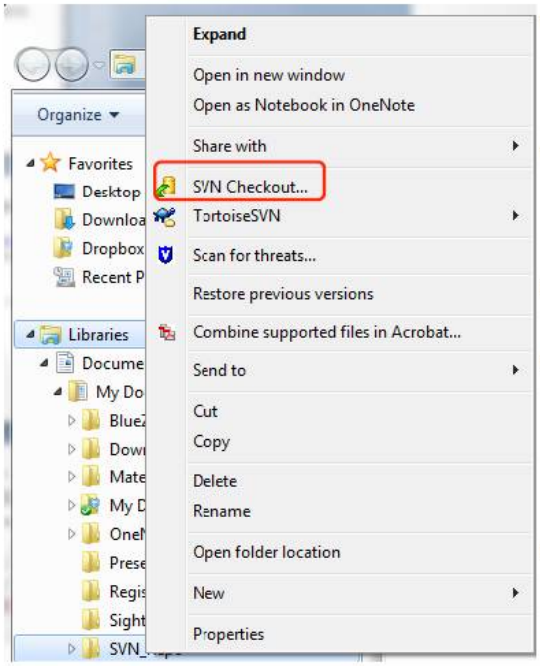
Create a local repository

- Create a new directoy somewhere (perhaps under "My Documents"). I called mine "SVN_Repo."
- Right-click on the SVN_Repo folder. The menu that pops up will have a couple of new items: the "SVN Checkout..." and "TortoiseSVN." Choose "TortoiseSVN."
- Click on "Create repository here" from the menu that pops up.



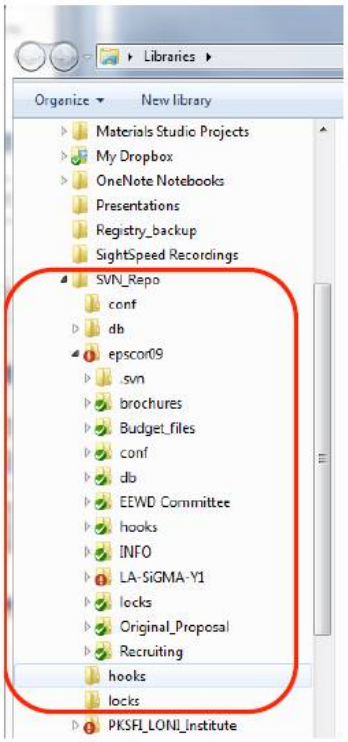


Do the first "Check Out"



- Once the repository is successfully created, right-click on the "SVN_Repo" directory again, and choose "SVN Checkout..."
- The program will ask you for the repo address. For the LA-SiGMA project, it is https://svn.cct.lsu.edu/repos/proposals/epscor09
- Enter the user-id and password provided by CCT.
- The checkout process now starts, and creates an exact replica of the SVN file system on your local drive. You should do this step while connected to a fast connection, because our repo has become quite large.
- You should allow a fair amout of time. [I did this at home, on a 54 Mbps wireless, and it took about 2.5 hours.]

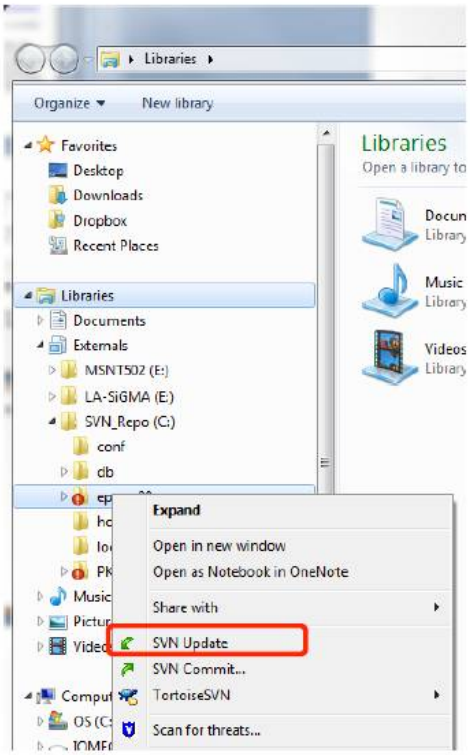
Directory Structure



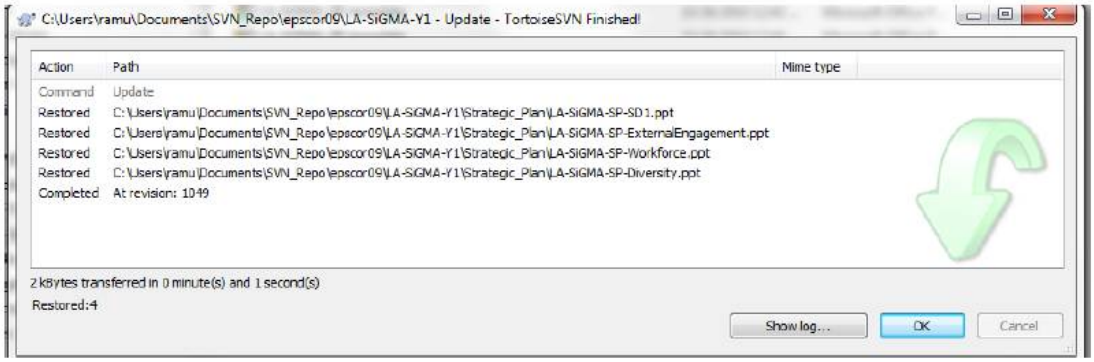
- After a successful checkout, you will see that several folders have appeared under "SVN_Repo." Mine is shown on left.
- Note the red ! (exclamation) and the green check marks appearing on the directory icons.
- The red symbol on the top level directory (epscor09) means that one or more of the subdirectories contain a file that I have changed, making my local version different from the copy on the SVN.
- The green check marks indicate that all the files in those folders have not been changed locally since the last update.

Working with the SVN: Update and Commit

- When working with the SVN on a daily basis, there are just two things to remember: update, and commit.
- Before you start working on any file in the SVN, it is important to make sure you have the most up-to-date versions of all files. For this, you would do an "update" from the SVN menu: right click on the directory icon, and choose "SVN Update."
- The update process copies only the files that have changed since the last update.
- The commit process copies files you may have changed locally to the central SVN (so that others can get it by updating).



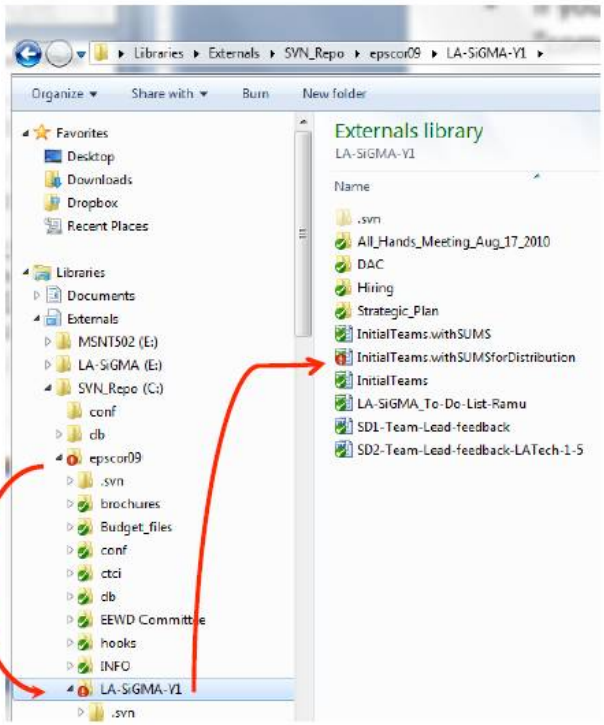
The screen-shot of an update process



I had deliberately deleted four files from my local SVN repository prior to this update. You can see that the update process restored those files from the central SVN. Note that we are now at revision # 1049.

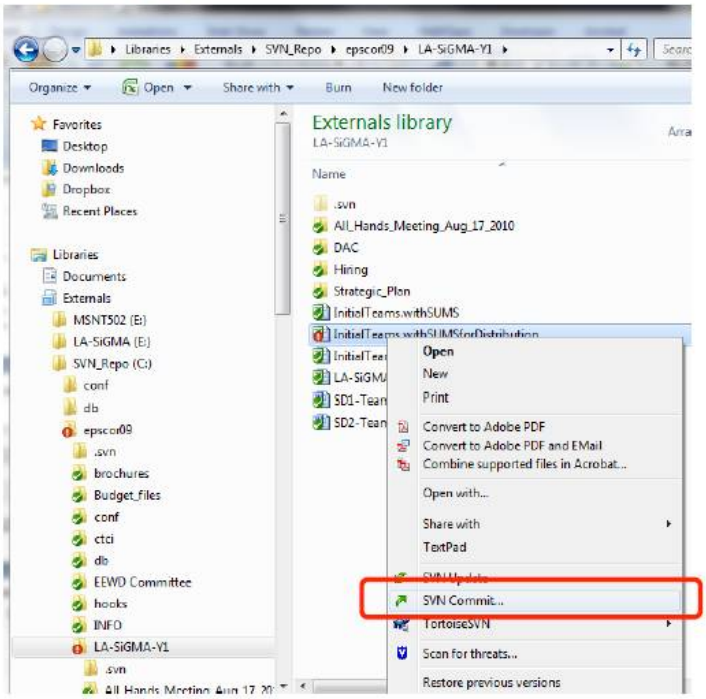
Committing changes - 1

- If you make changes to a file, or if you create a new file, it needs to be "committed" (not in the psychiatric sense in most cases) so that others can see it.
- In the screen shot to the right, I have deliberately modified one of the files. Note that it is marked by a red exclamation mark all the way up the directory tree to the top level.
- Note that all the other files (and sub-folder) have green check marks, indicating that their contents are consistent with the central SVN.
- If you create a new file, it will not have any mark on it, which is a clue that the central SVN knows nothing about that file's existence.



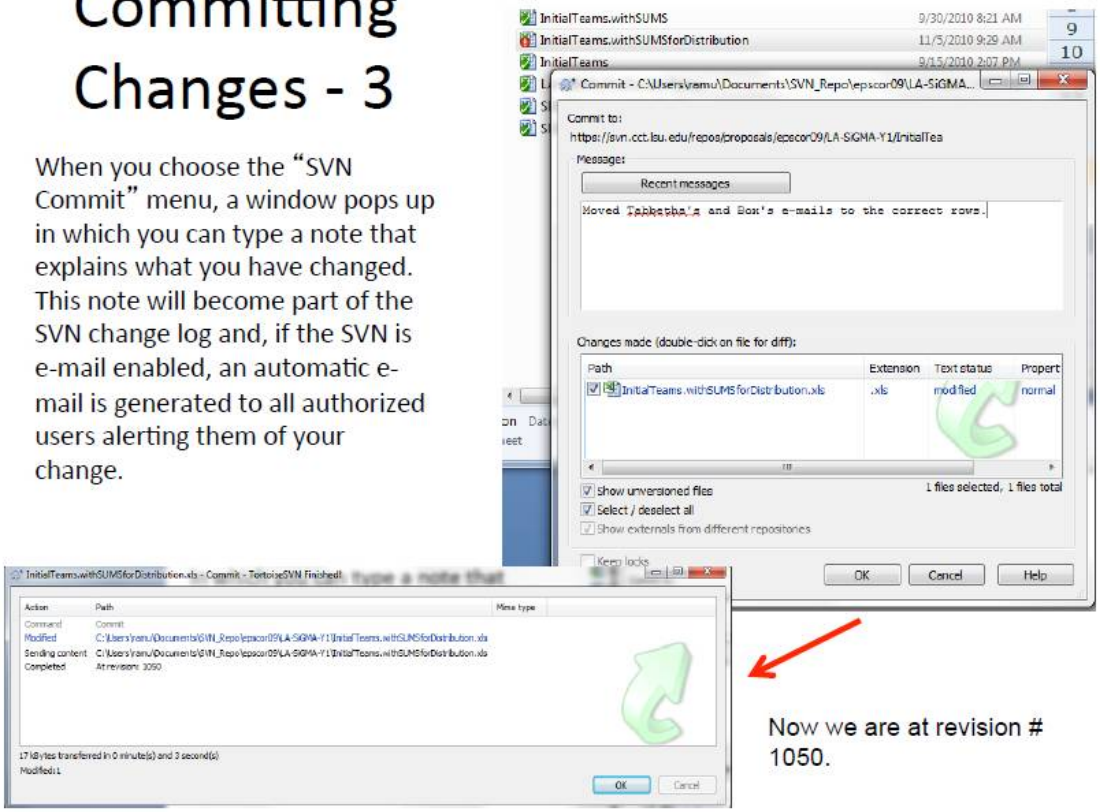
Committing Changes - 2

The screen shot shows that I am about to commit the file that I changed.



Committing Changes - 3

When you choose the "SVN Commit" menu, a window pops up in which you can type a note that explains what you have changed. This note will become part of the SVN change log and, if the SVN is e-mail enabled, an automatic e-mail is generated to all authorized users alerting them of your change.

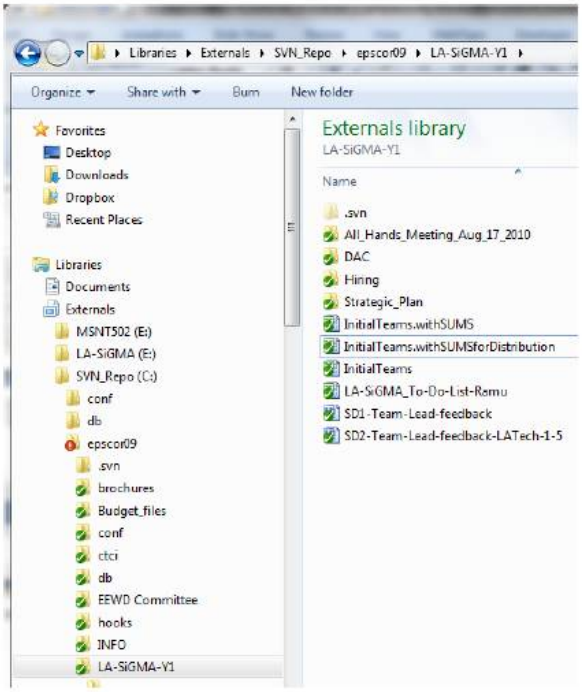


Now we are at revision # 1050.

One more slide on "commit"

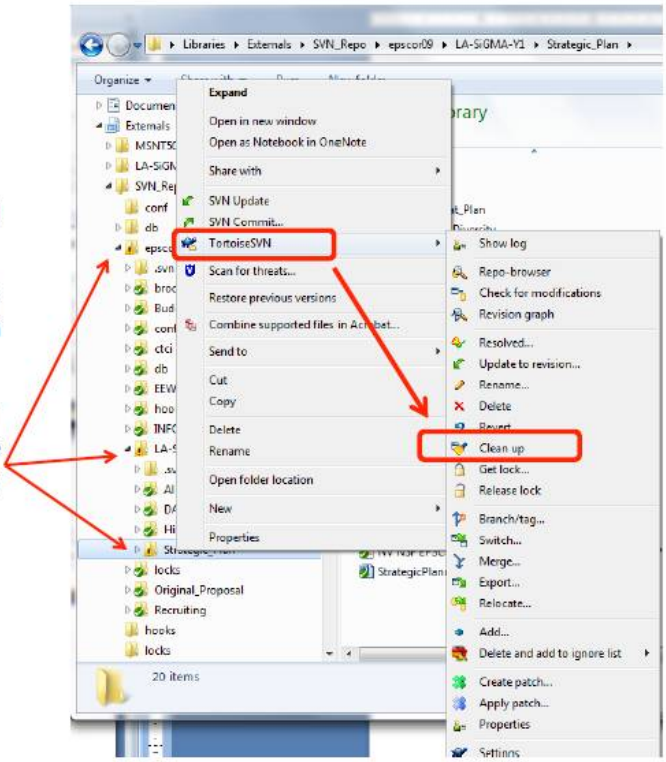
If all goes well with the "commit," when I refresh my directory, the red exclamation mark will change to a green check mark, telling me that I have achieved a harmonious union with the great mother SVN.

You may note that there is still a red exclamation mark on the high level directory "epscor09." This never goes away on my desktop machine. I think it is an artifact of my 64-bit client not interacting with the central SVN. I run the 32-bit client application on my laptop and it does not have this problem.



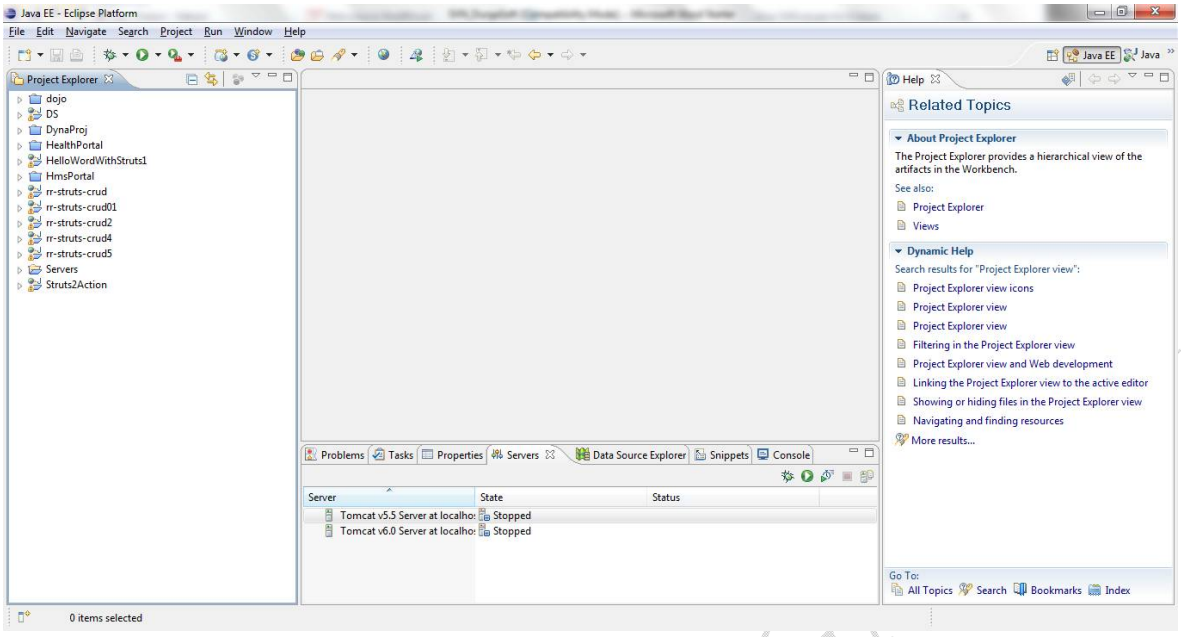
Clean up

- Sometimes, things go wrong, and conflicts arise.
- One way to resolve the conflict is to use the "Clean up" menu from "TortoiseSVN."
- If that does not work, you should copy the conflicted files (note the yellow triangle marks) to another directory, delete them from the local SVN Repo, and do an "Update."

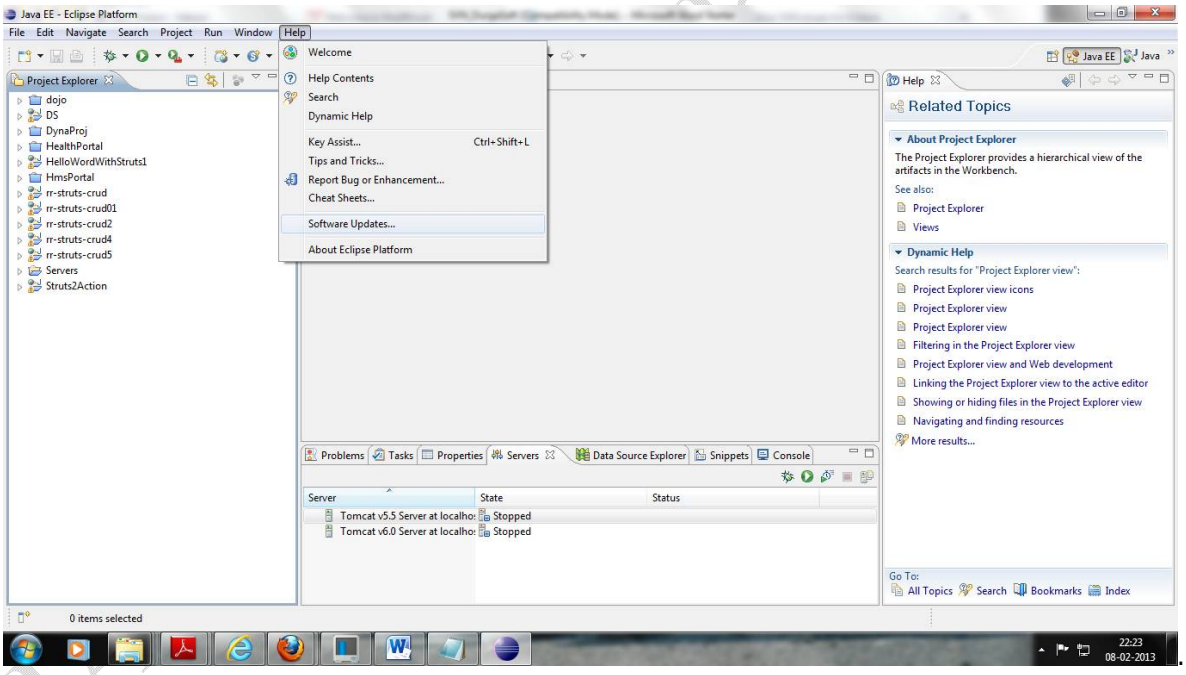


Procedure to configure SVN Plugin into Eclipse IDE :

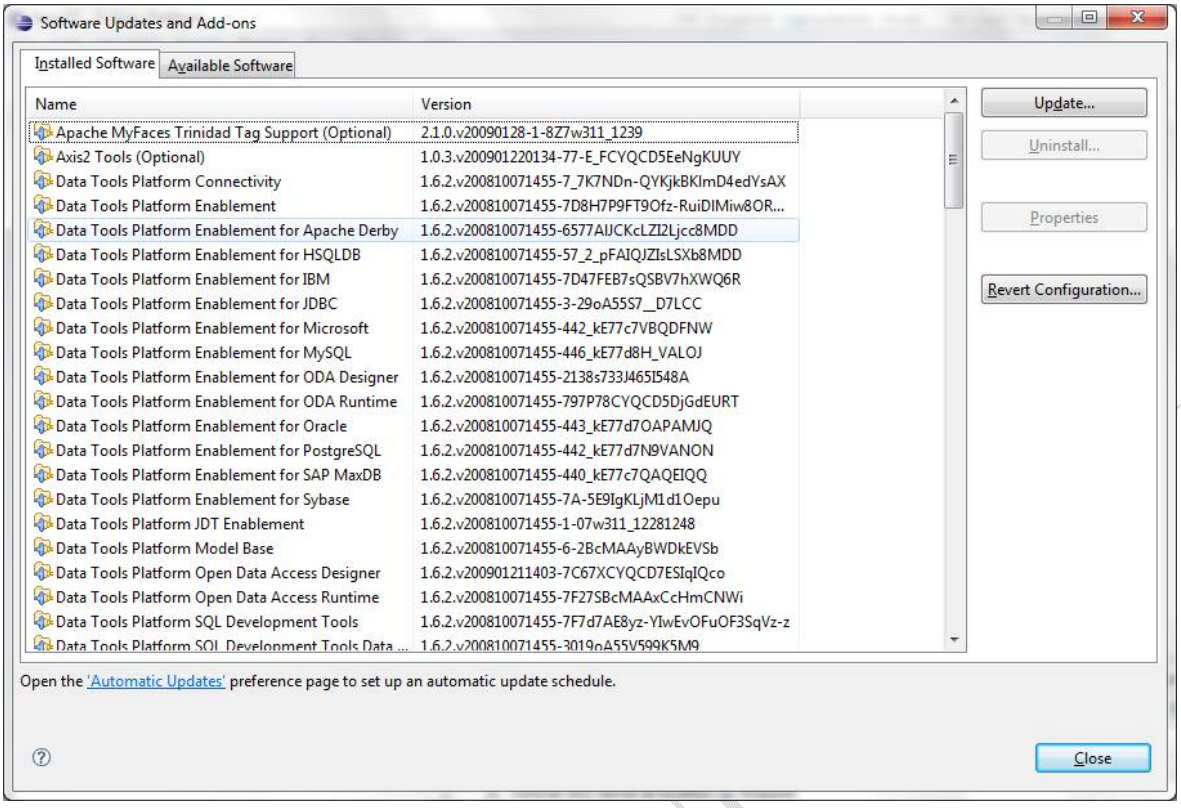
Step-1 : Launch Eclipse having new work space for programmer.



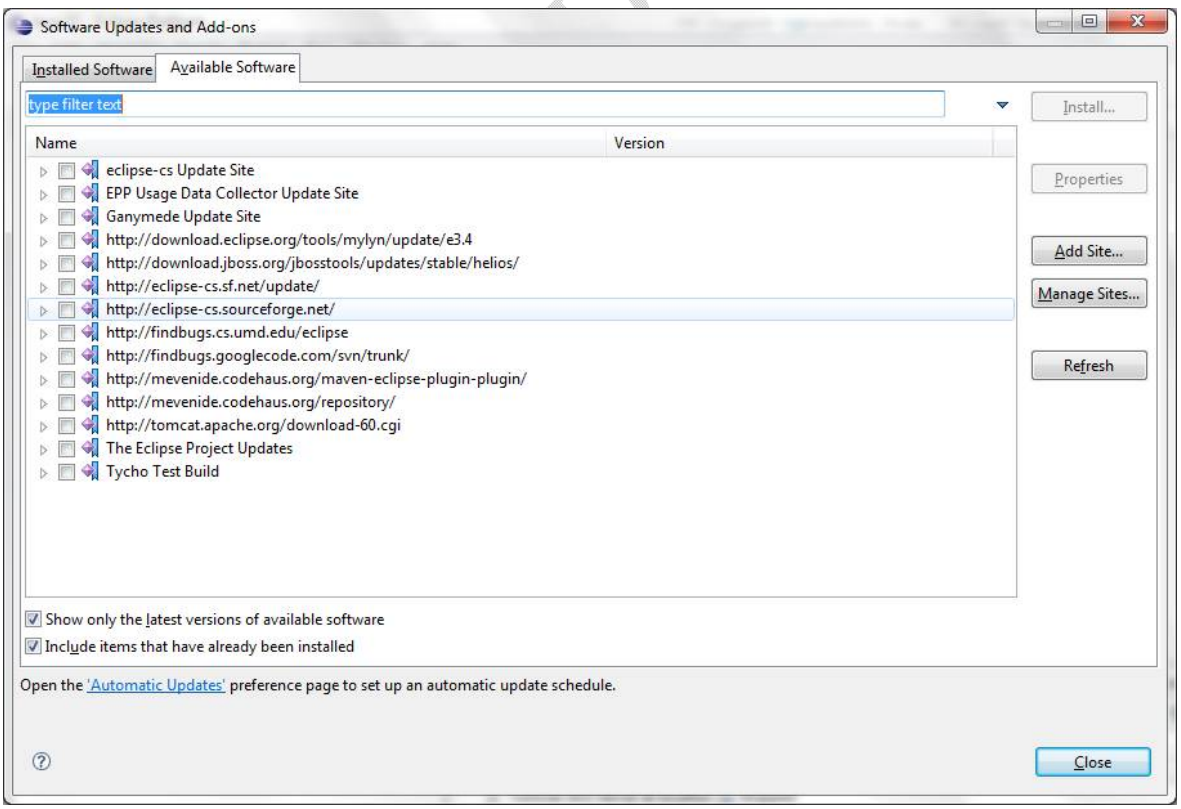
Step-2 : Click on Software Updates under Help



Step-3:

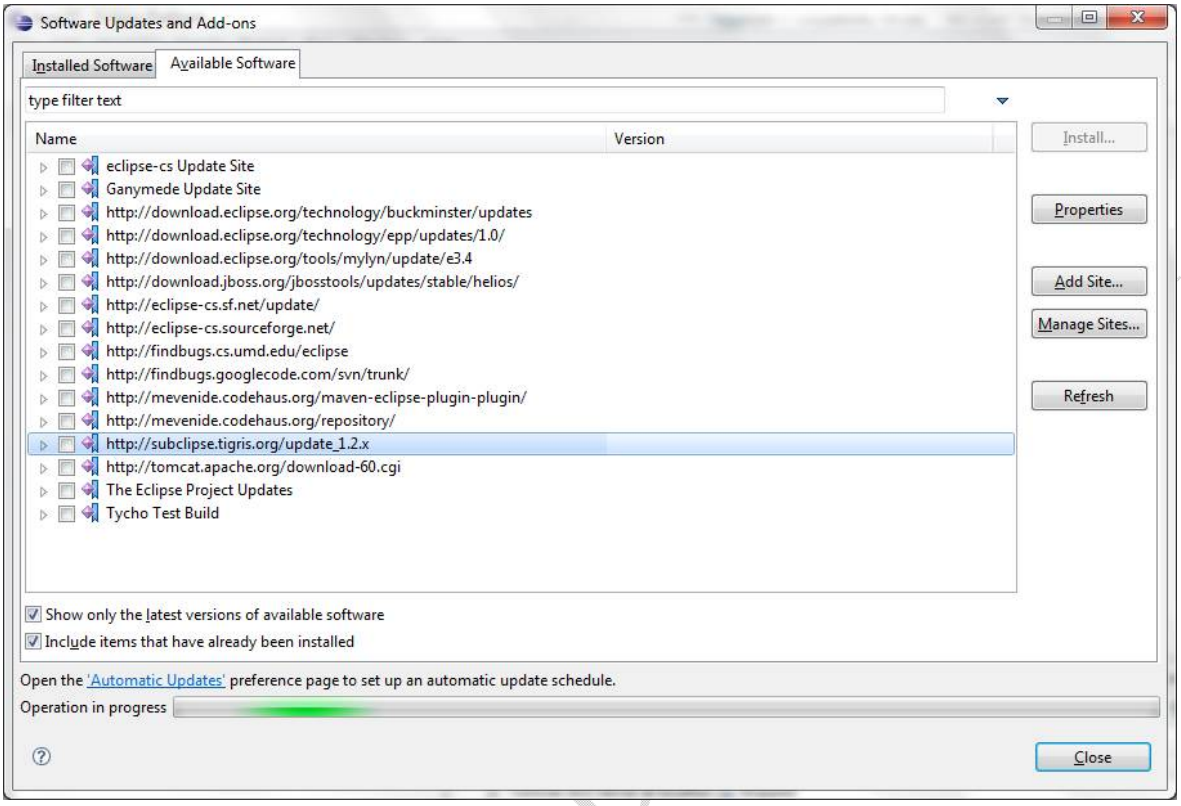

Software Updates and Add-ons
Installed Software
Available Software
Name
Version
Apache MyFaces Trinidad Tag Support (Optional) 2.1.0.v20090128-1-8Z7w311_1239
Axis2 Tools (Optional) 1.0.3.v200901220134-77-E_FCYQCD5EeNgKUUY
Data Tools Platform Connectivity 1.6.2.v200810071455-7_7K7NDn-QYKjkBKImD4edYsAX
Data Tools Platform Enablement 1.6.2.v200810071455-7D8H7P9FT9Ofz-RuiDIMiw8OR...
Data Tools Platform Enablement for Apache Derby 1.6.2.v200810071455-6577AIJCKcLZI2Ljcc8MDD
Data Tools Platform Enablement for HSQLDB 1.6.2.v200810071455-57_2_pFAIQJZIsLSXb8MDD
Data Tools Platform Enablement for IBM 1.6.2.v200810071455-7D47FEB7sQSBV7hXWQ6R
Data Tools Platform Enablement for JDBC 1.6.2.v200810071455-3-29oA55S7__D7LCC
Data Tools Platform Enablement for Microsoft 1.6.2.v200810071455-442_kE77c7VBQDFNW
Data Tools Platform Enablement for MySQL 1.6.2.v200810071455-446_kE77d8H_VALOJ
Data Tools Platform Enablement for ODA Designer 1.6.2.v200810071455-2138s733J4651548A
Data Tools Platform Enablement for ODA Runtime 1.6.2.v200810071455-797P78CYQCD5DjGdEURT
Data Tools Platform Enablement for Oracle 1.6.2.v200810071455-443_kE77d7OAPAMJQ
Data Tools Platform Enablement for PostgreSQL 1.6.2.v200810071455-442_kE77d7N9VANON
Data Tools Platform Enablement for SAP MaxDB 1.6.2.v200810071455-440_kE77c7QAQEIQQ
Data Tools Platform Enablement for Sybase 1.6.2.v200810071455-7A-5E9IgKLjM1d1Oepu
Data Tools Platform JDT Enablement 1.6.2.v200810071455-1-07w311_12281248
Data Tools Platform Model Base 1.6.2.v200810071455-6-2BcMAAyBWDkEVSb
Data Tools Platform Open Data Access Designer 1.6.2.v200901211403-7C67XCYQCD7ESIqIQco
Data Tools Platform Open Data Access Runtime 1.6.2.v200810071455-7F27SBcMAAxCcHmCNWi
Data Tools Platform SQL Development Tools 1.6.2.v200810071455-7F7d7AE8yz-YIwEvOFuOF3SqVz-z
Update...
Uninstall...
Properties
Revert Configuration...
Open the 'Automatic Updates' preference page to set up an automatic update schedule.
Close

Step-4:

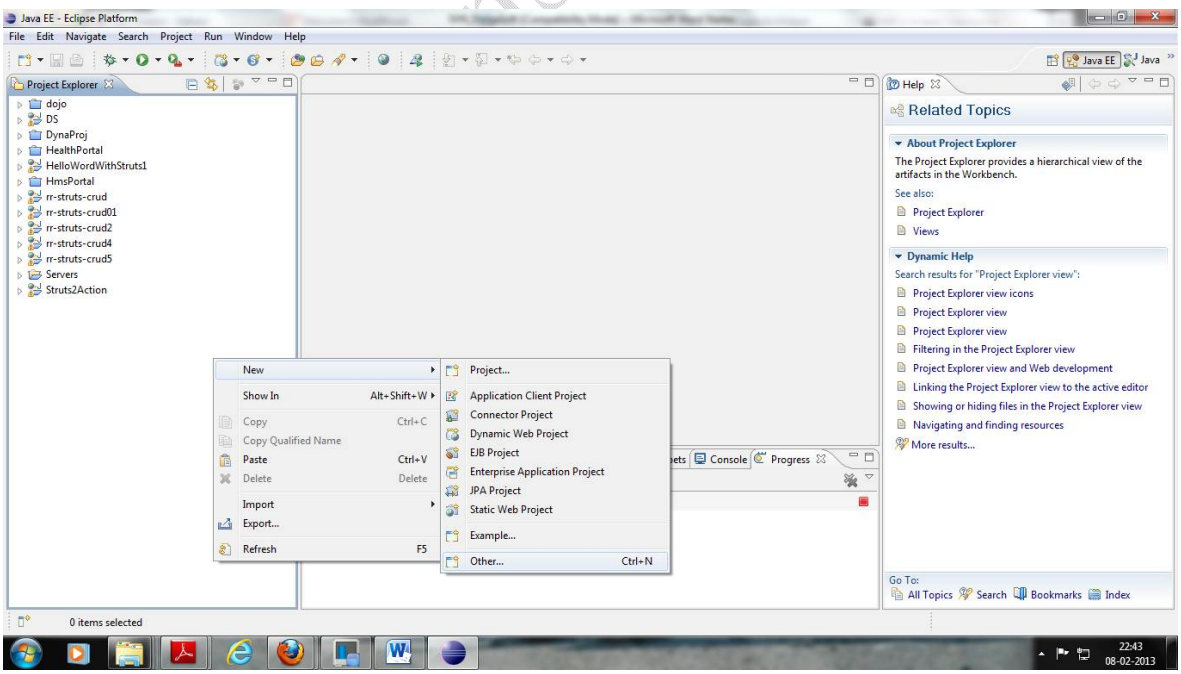

Software Updates and Add-ons
Installed Software
Available Software
type filter text
Name
Version
eclipse-cs Update Site
EPP Usage Data Collector Update Site
Ganymede Update Site
http://download.eclipse.org/tools/mylyn/update/e3.4
http://download.jboss.org/jbosstools/updates/stable/helios/
http://eclipse-cs.sf.net/update/
http://eclipse-cs.sourceforge.net/
http://findbugs.cs.umd.edu/eclipse
http://findbugs.googlecode.com/svn/trunk/
http://mevenide.codehaus.org/maven-eclipse-plugin-plugin/
http://mevenide.codehaus.org/repository/
http://tomcat.apache.org/download-60.cgi
The Eclipse Project Updates
Tycho Test Build
Install...
Properties
Add Site...
Manage Sites...
Refresh
Show only the latest versions of available software
Include items that have already been installed
Open the 'Automatic Updates' preference page to set up an automatic update schedule.
Close

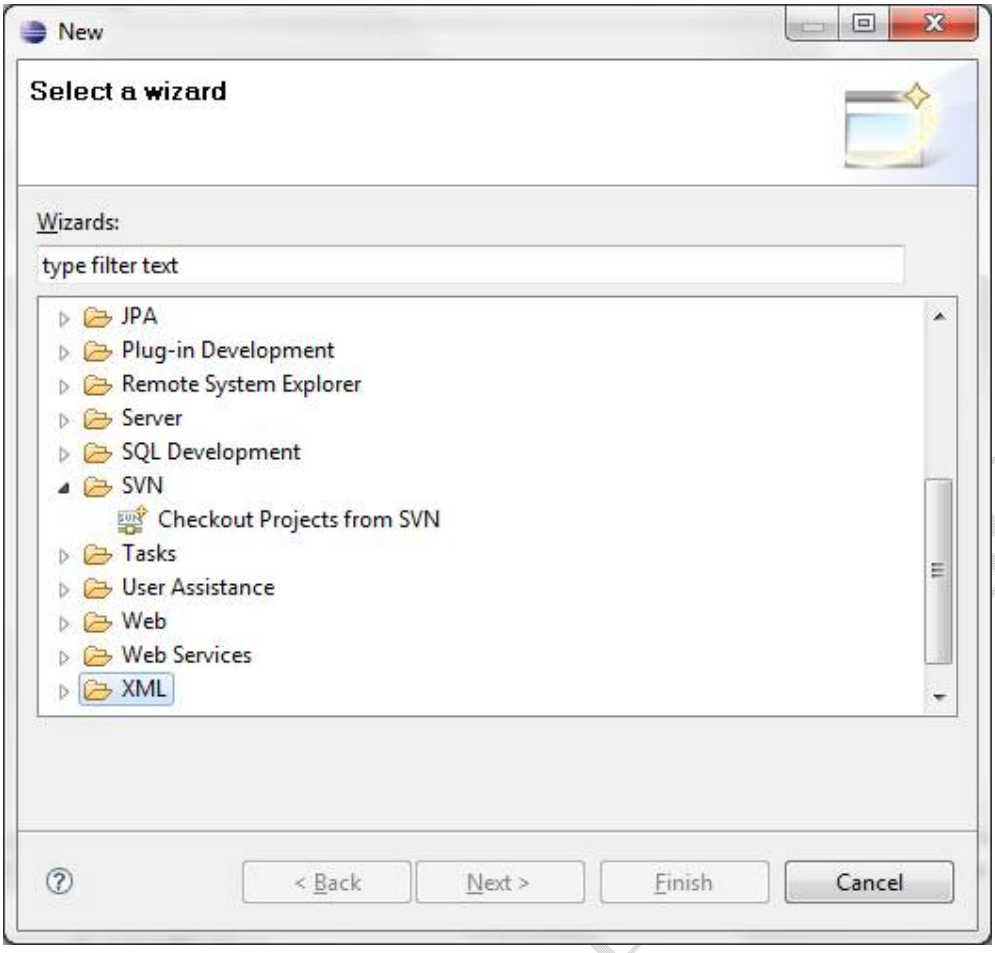
Step-5:

Click on Add Site and enter http://subclipse.tigris.org/update_1.2.x url and click on install. After installing the SVN and restart the Elipse.

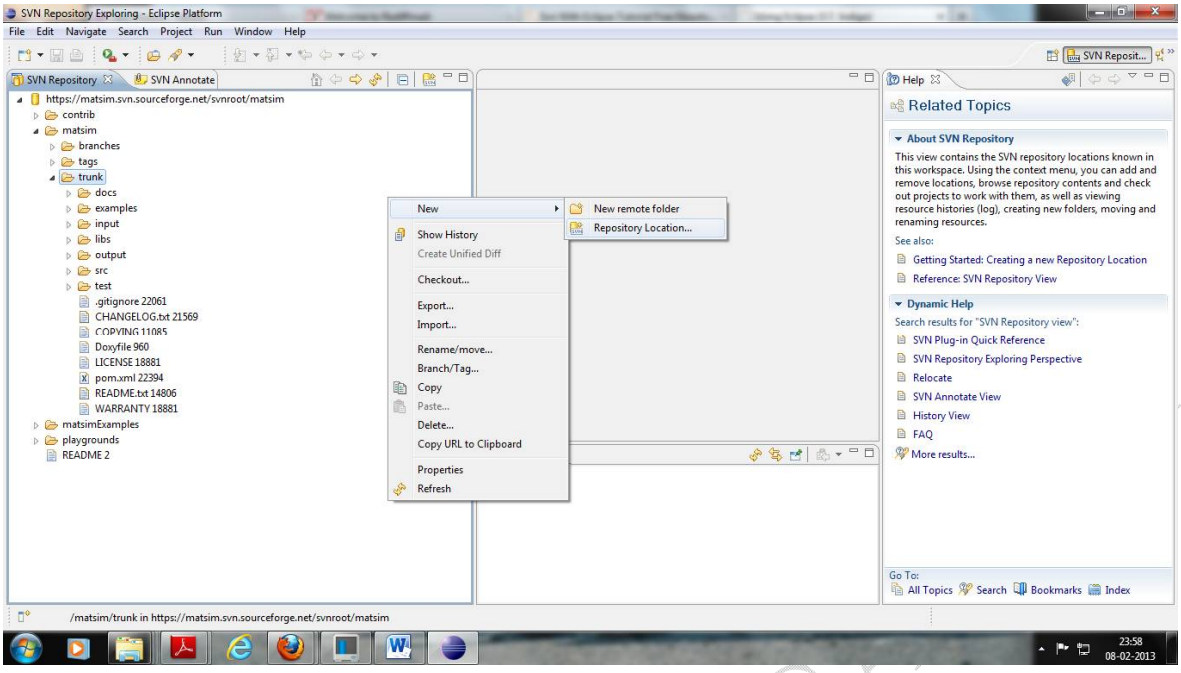


Step-6: To confirm SVN is configured

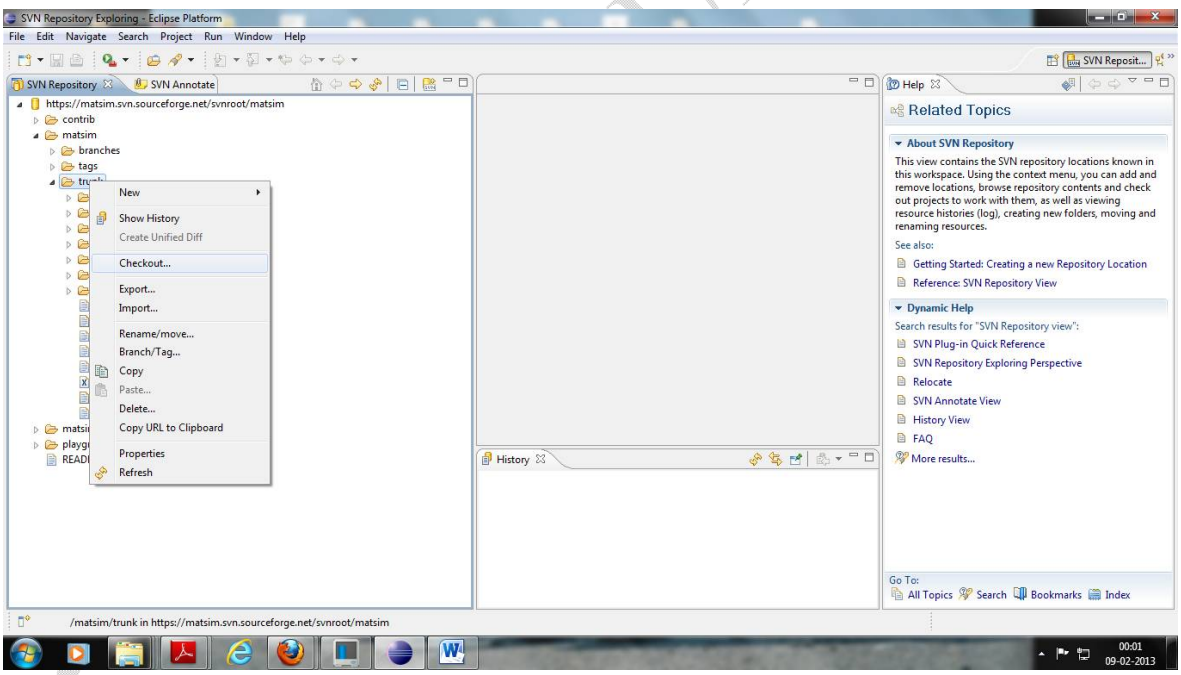



New
Select a wizard
Wizards:
type filter text
JPA
Plug-in Development
Remote System Explorer
Server
SQL Development
SVN
Checkout Projects from SVN
Tasks
User Assistance
Web
Web Services
XML
< Back
Next >
Finish
Cancel

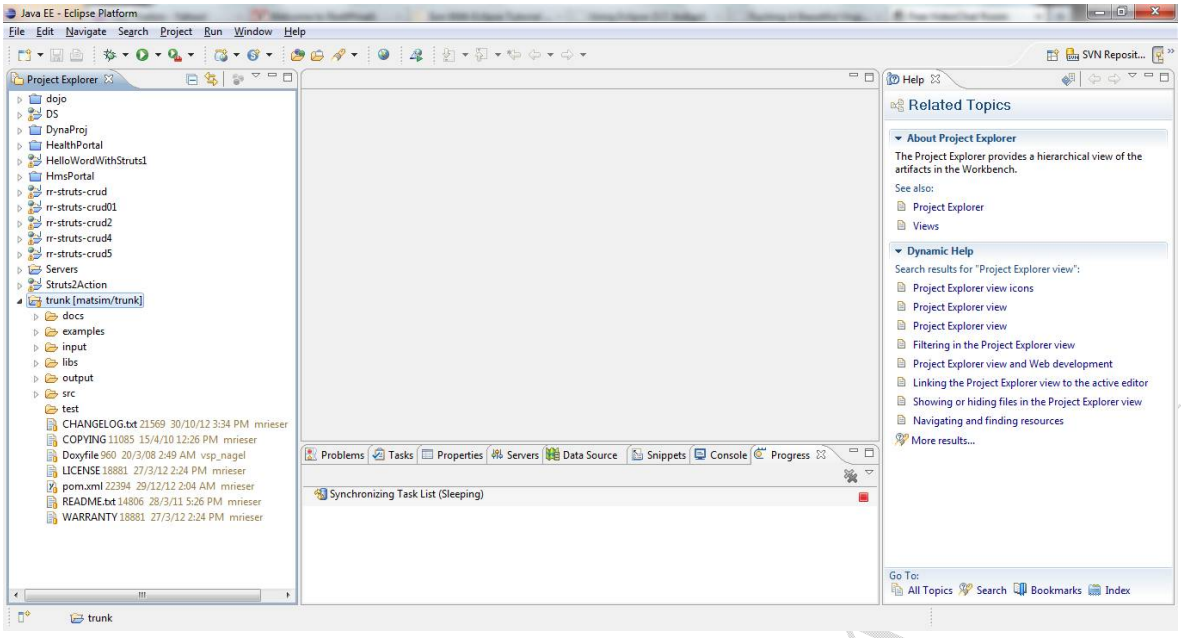

FREE TRAINING VIDEOS
YouTube
3000+
VIDEOS
www.youtube.com/durgasoftware

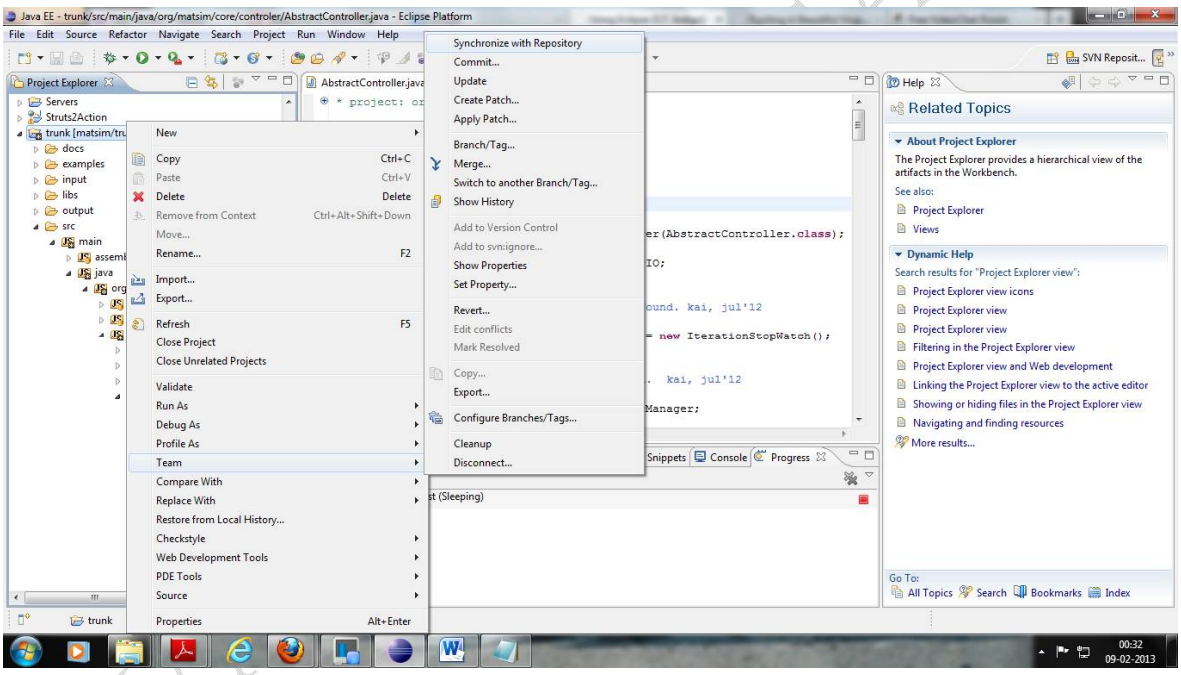


Check out : Check out code from Repository.



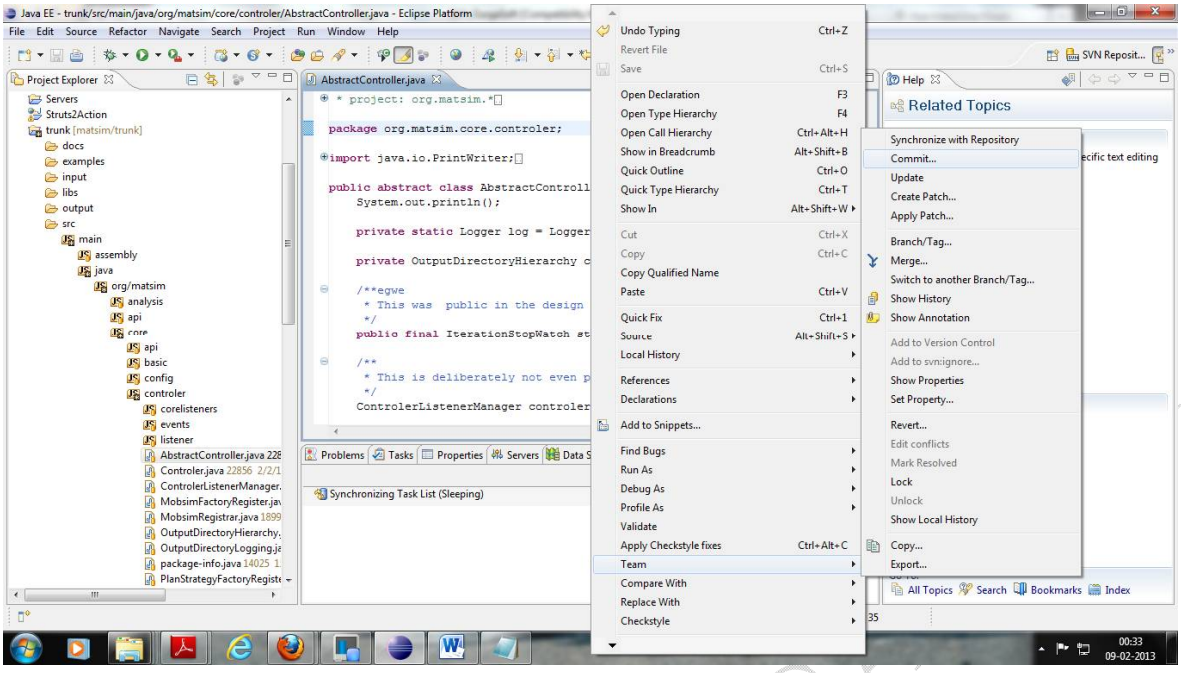
Checkout code from Repository

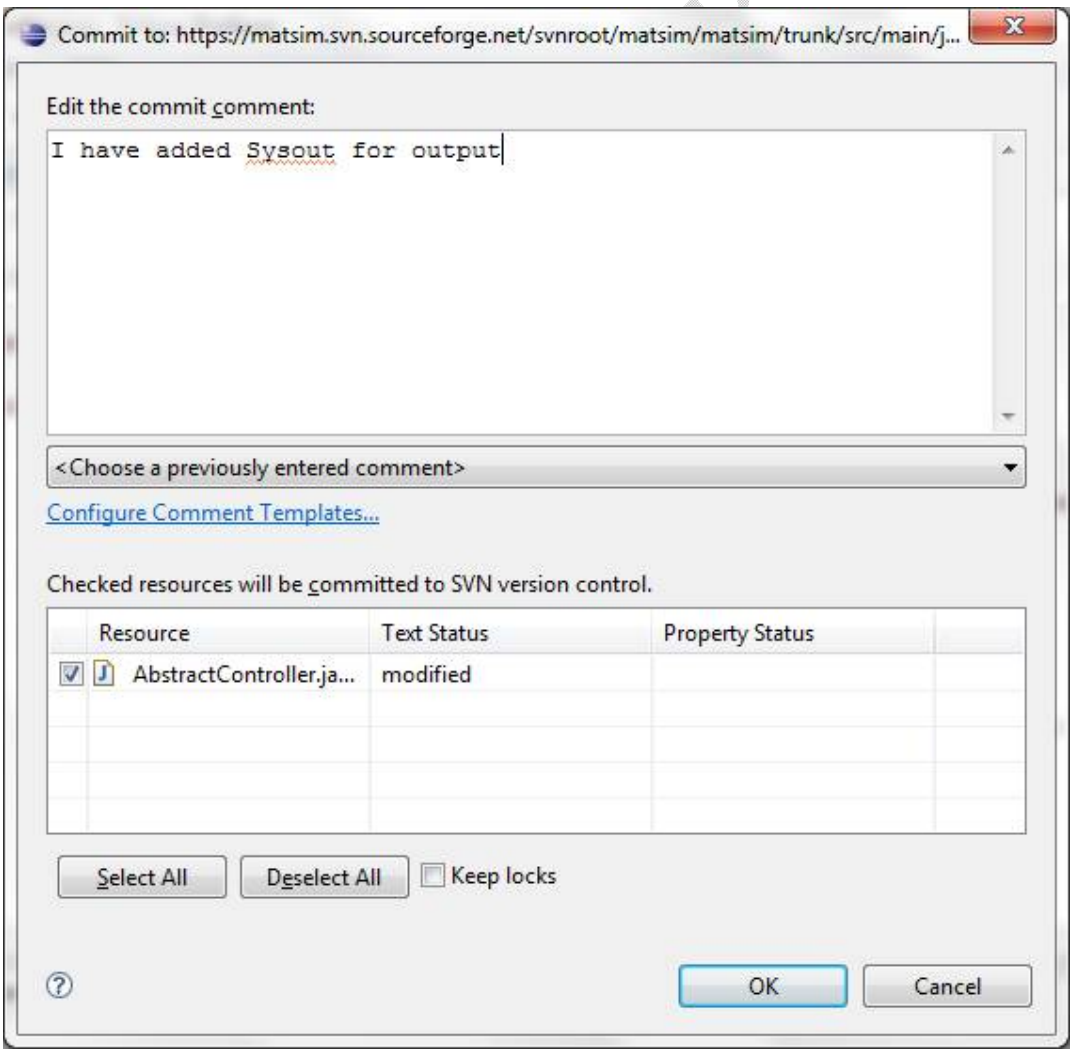




Check In:

Right click on AbstractController.java → team → commit (checkin) → enter some comment → finish.


Java EE - trunk/src/main/java/org/matsim/core/controler/AbstractController.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
AbstractController.java
Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Local History
References
Declarations
Add to Snippets...
Find Bugs
Run As
Debug As
Profile As
Validate
Apply Checkstyle fixes Ctrl+Alt+C
Team
Compare With
Replace With
Checkstyle
Synchronize with Repository
Commit...
Update
Create Patch...
Apply Patch...
Branch/Tag...
Merge...
Switch to another Branch/Tag...
Show History
Show Annotation
Add to Version Control
Add to svnignore...
Show Properties
Set Property...
Revert...
Edit conflicts
Mark Resolved
Lock
Unlock
Show Local History
Copy...
Export...
Help
Related Topics
All Topics Search Bookmarks Index
Synchronizing Task List (Sleeping)
00:33
09-02-2013


Commit to: https://matsim.svn.sourceforge.net/svnroot/matsim/matsim/trunk/src/main/j...
Edit the commit comment:
I have added Sysout for output
<Choose a previously entered comment>
Configure Comment Templates...
Checked resources will be committed to SVN version control.
Resource
Text Status
Property Status
AbstractController.ja...
modified
Select All
Deselect All
Keep locks
OK
Cancel

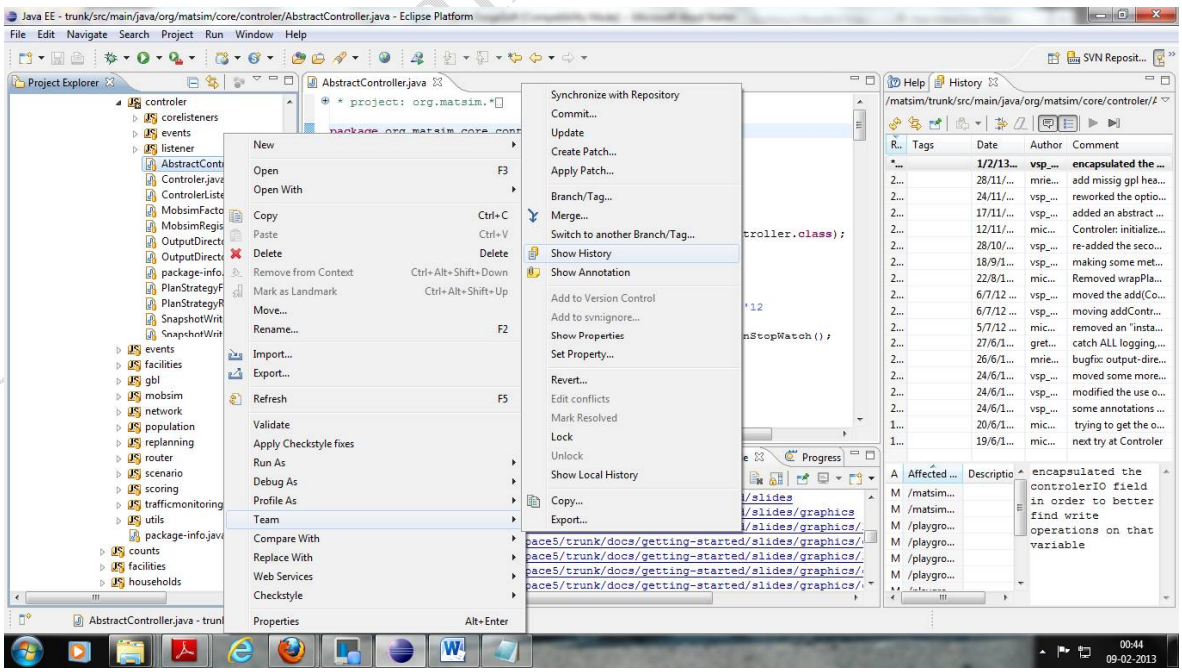
We have to write the comments for which purpose file is modified.

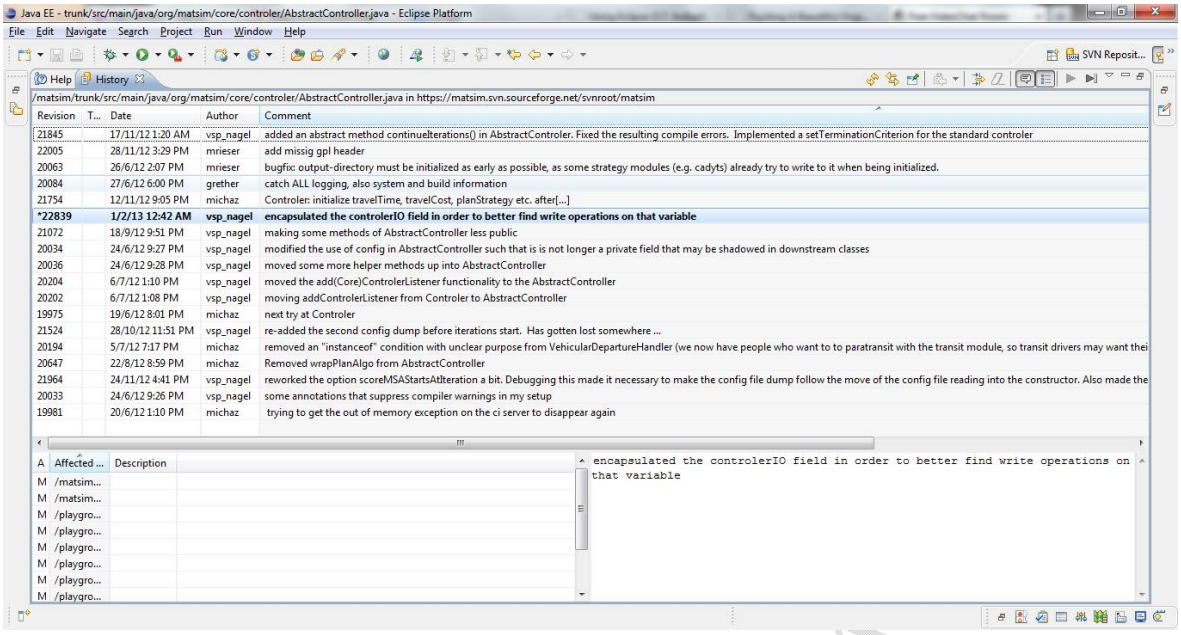
Click on OK. We have to provide user name and password then the latest changes will be updated into the repository.

- A new version will create for file when you perform commit operation.

History : To replace current file content with one of the existing old version of the cvs repository.

- Click on .java file → team → show history → Go to history window → double click on version what ever you want.



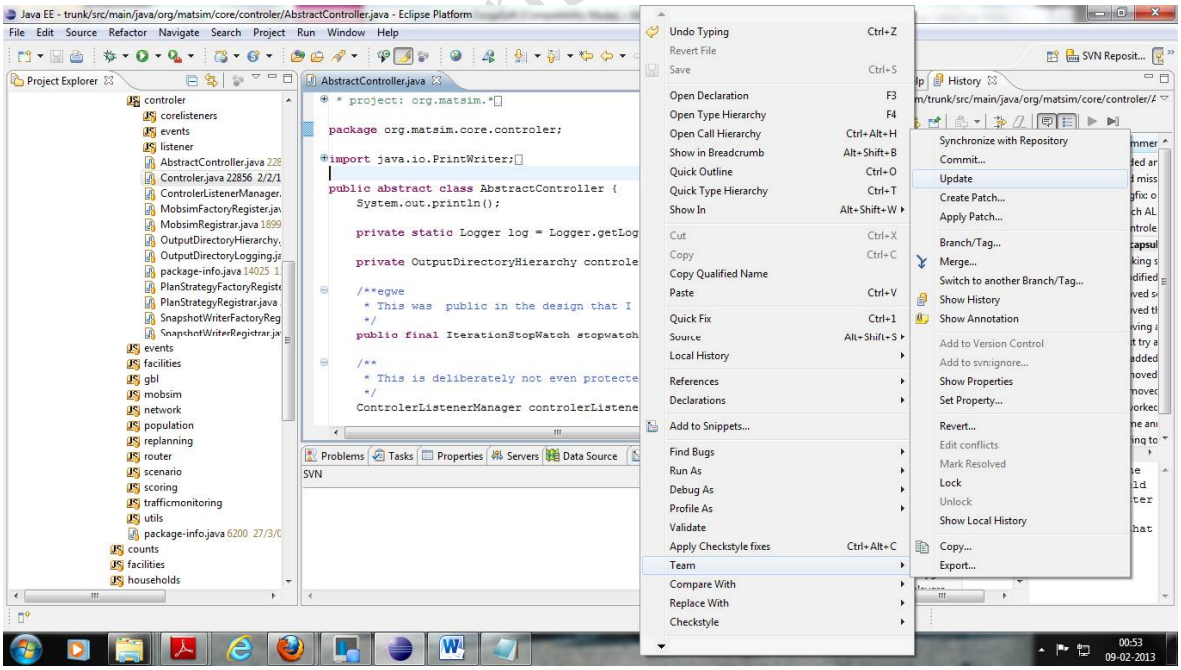


Update : This operation can be used for two purposes.

i) To get the updated version of the source file.

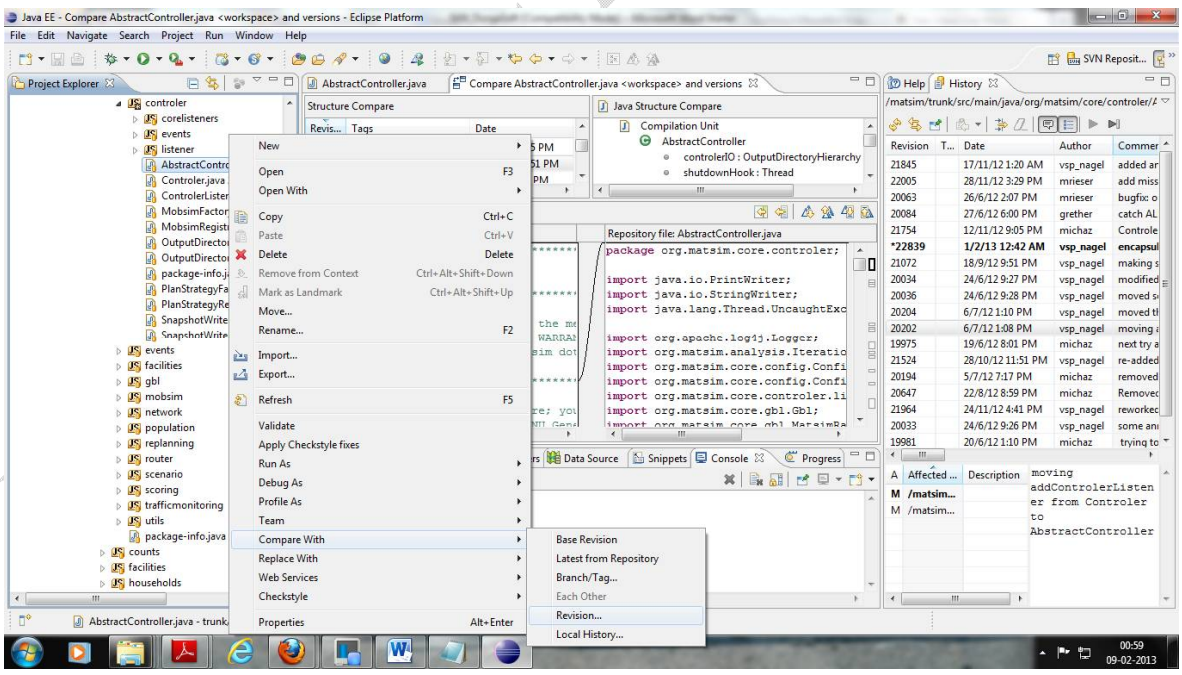
ii) To update the content of current file in cvs repository.

" Right click on file → team → update. "



Compare : To compare code of current version with the different versions in repository.

" Right click on source file → compare with → Revision → Go to Revision window select a version for compare "



Compare : To compare code of current version with the latest version in repository.

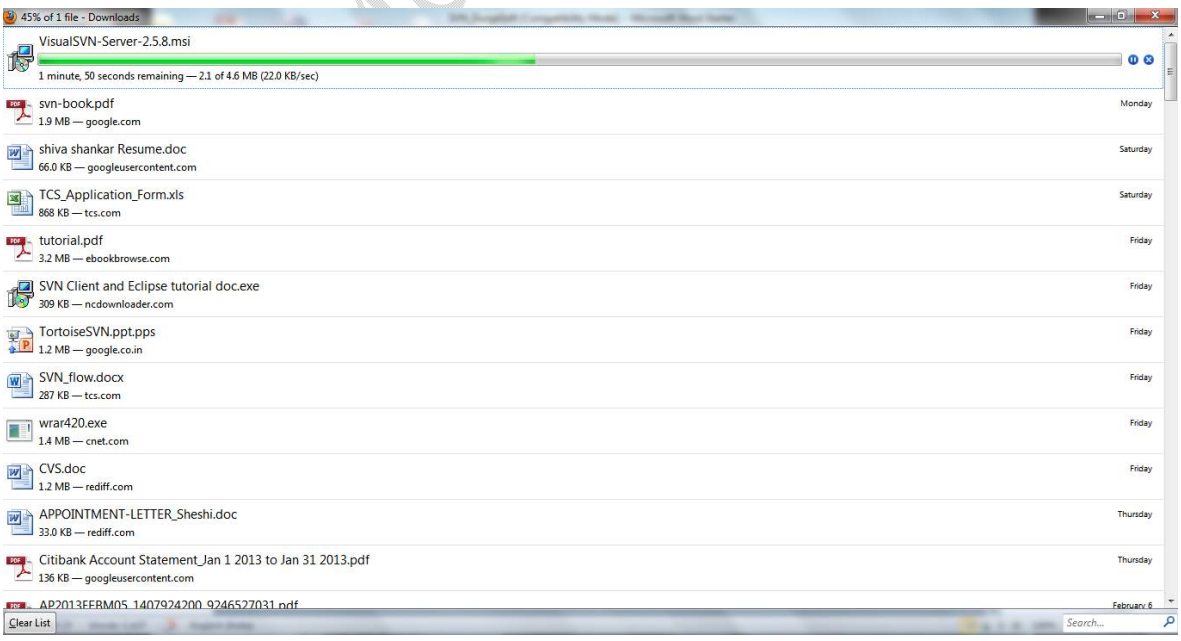
"Right click on source file → compare with →Latest From Repository" for compare.

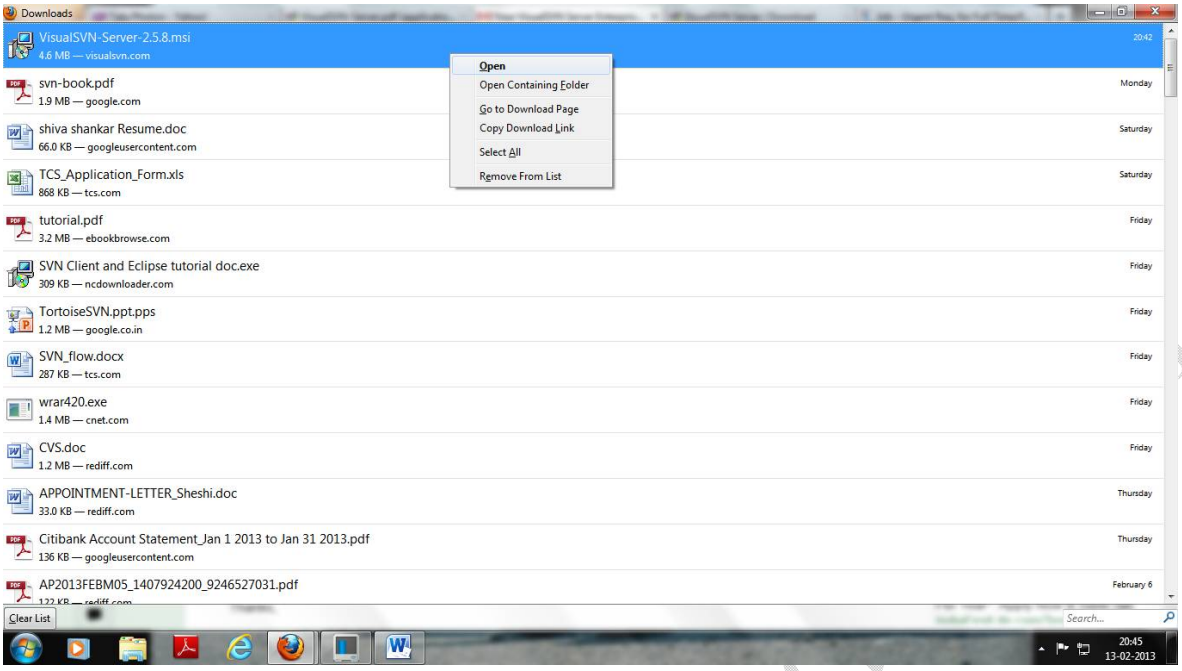
Compare : To compare code of current version with the Base version in repository.

"Right click on source file → compare with →Base Revision" for compare.

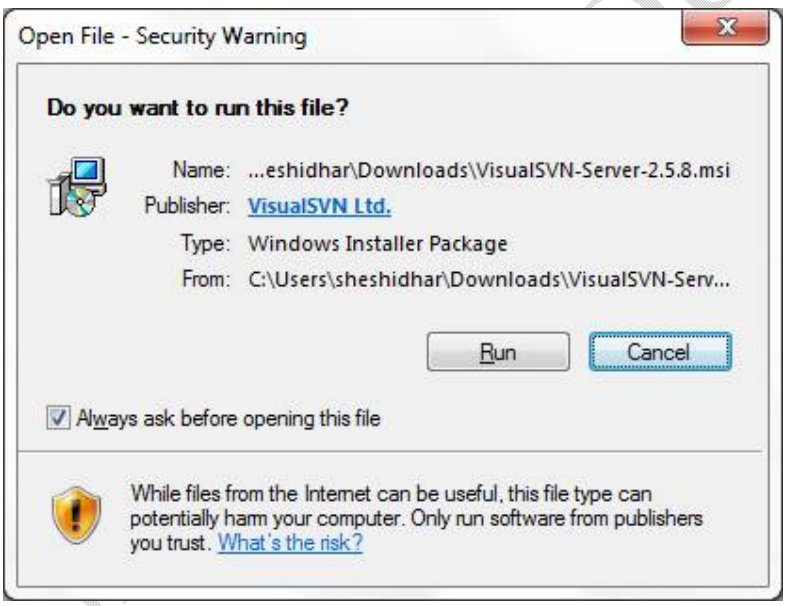
SVN Server Windows Installation:

Download VisualSVN Server from the below url:http://www.visualsvn.com/server/download/

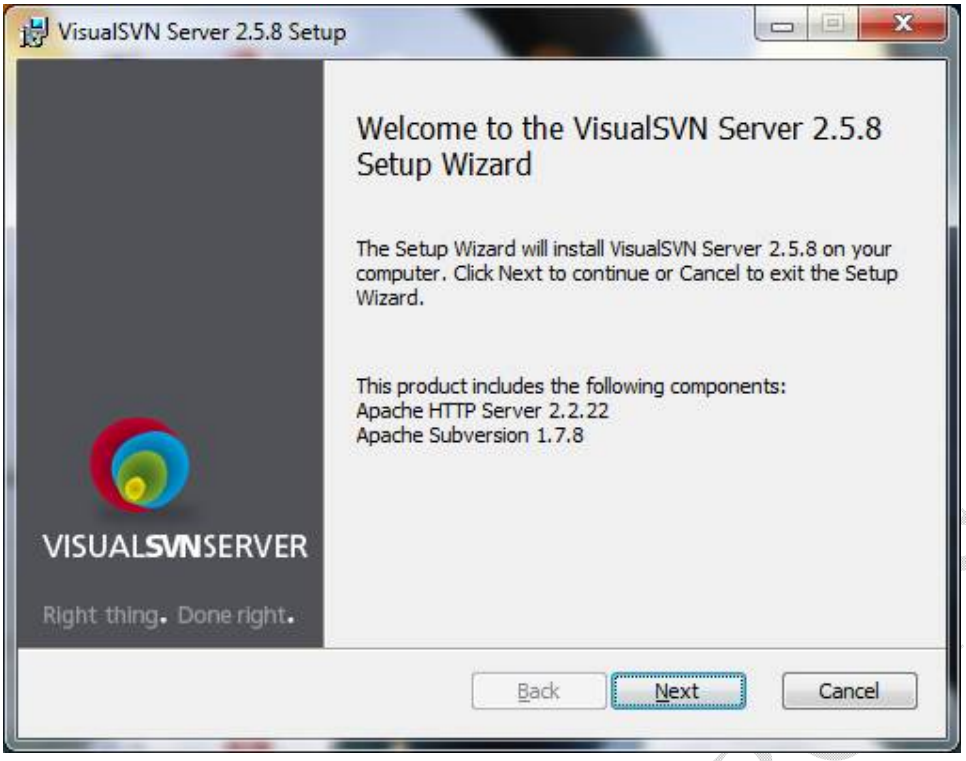




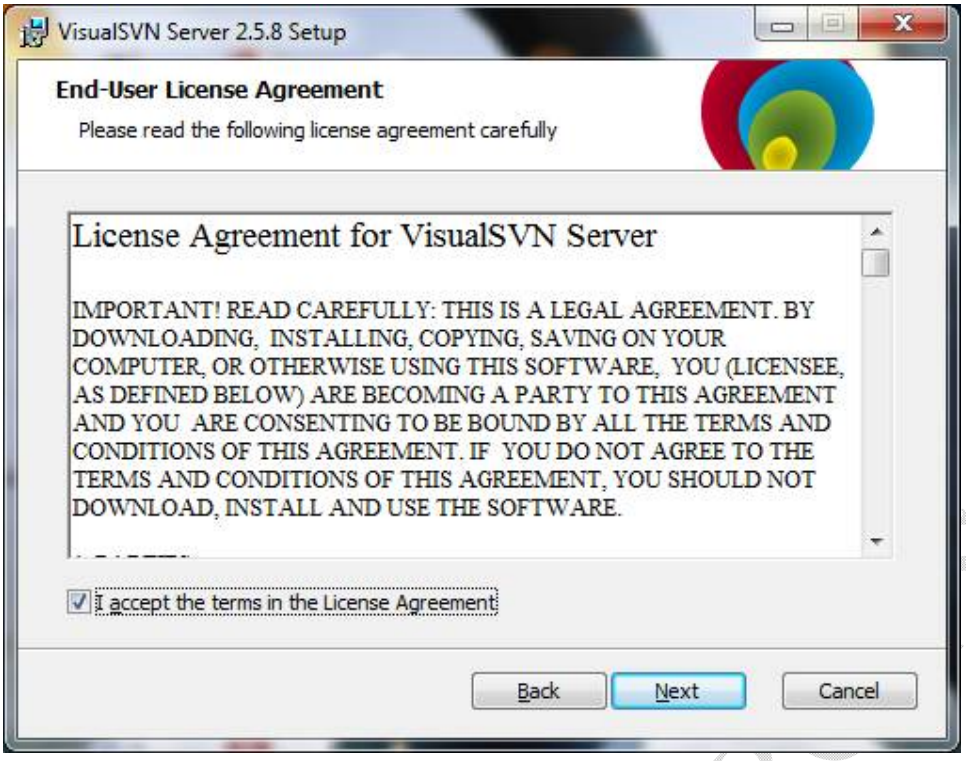
Right click on VisualServer.exe and Right click and click on Install



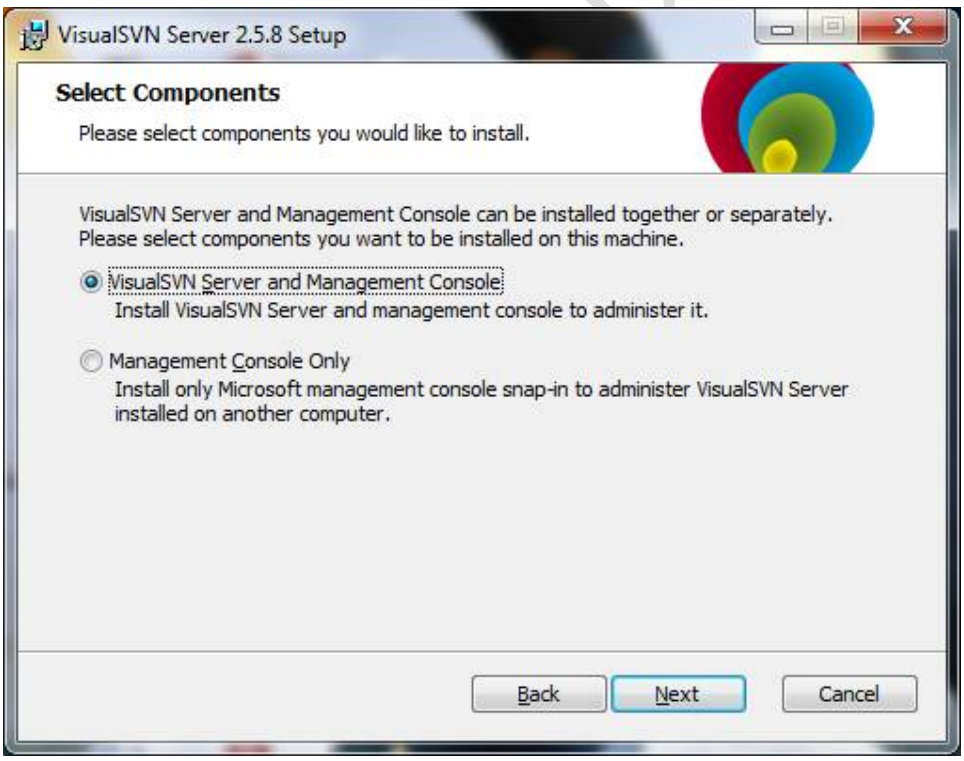
Click on Run



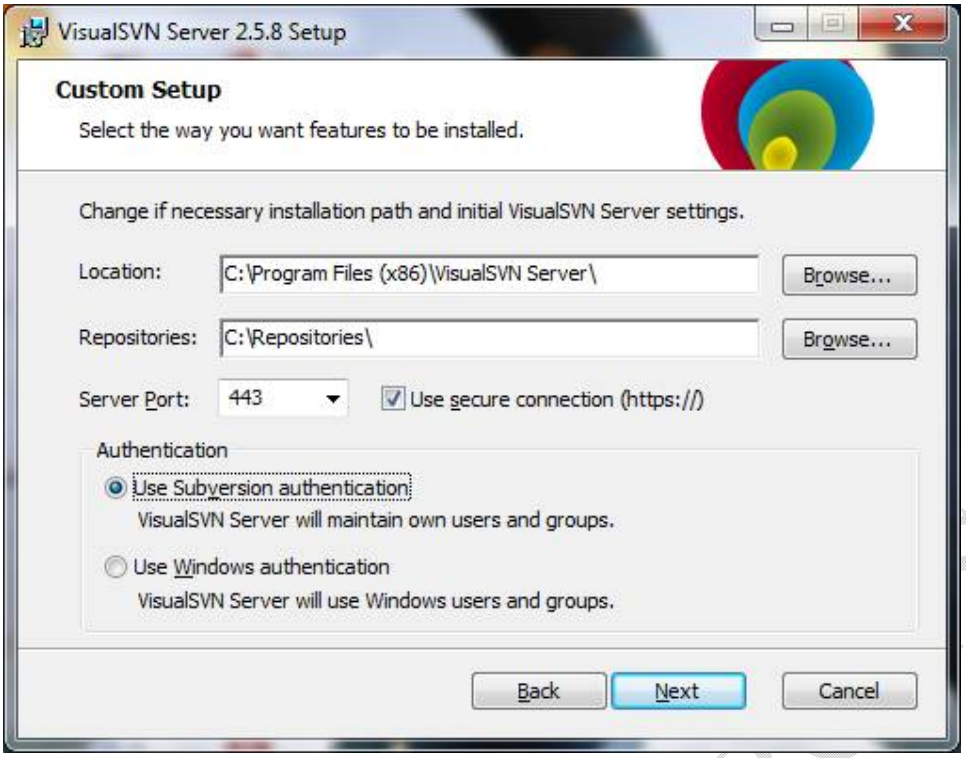
Click on Next



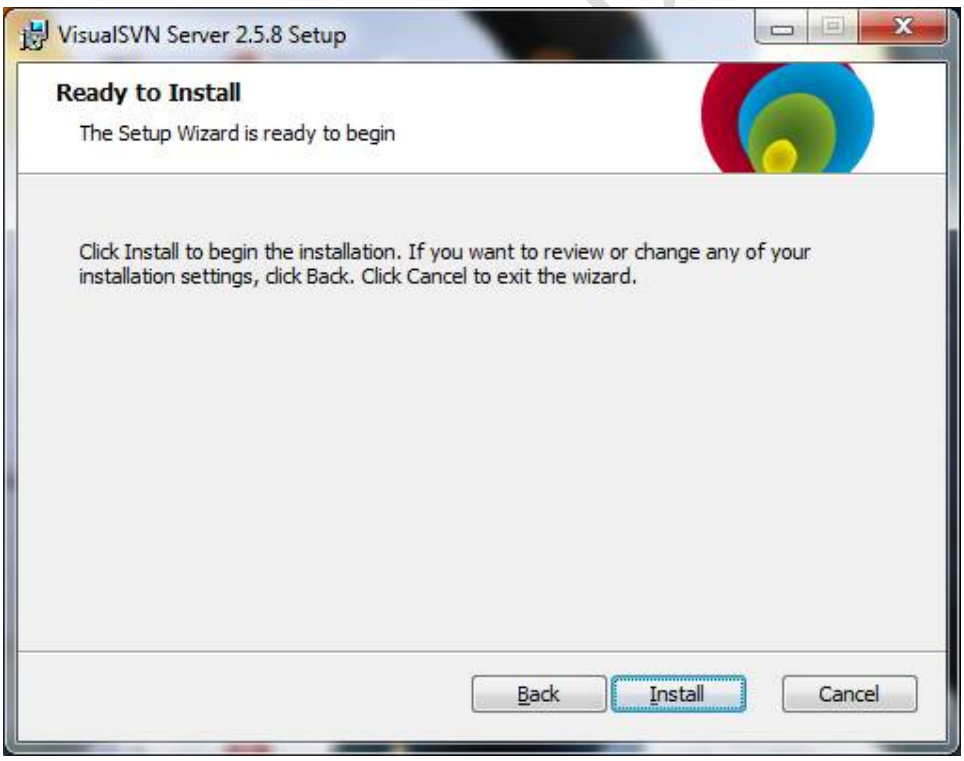
Select accept the license Agreement and click Nex



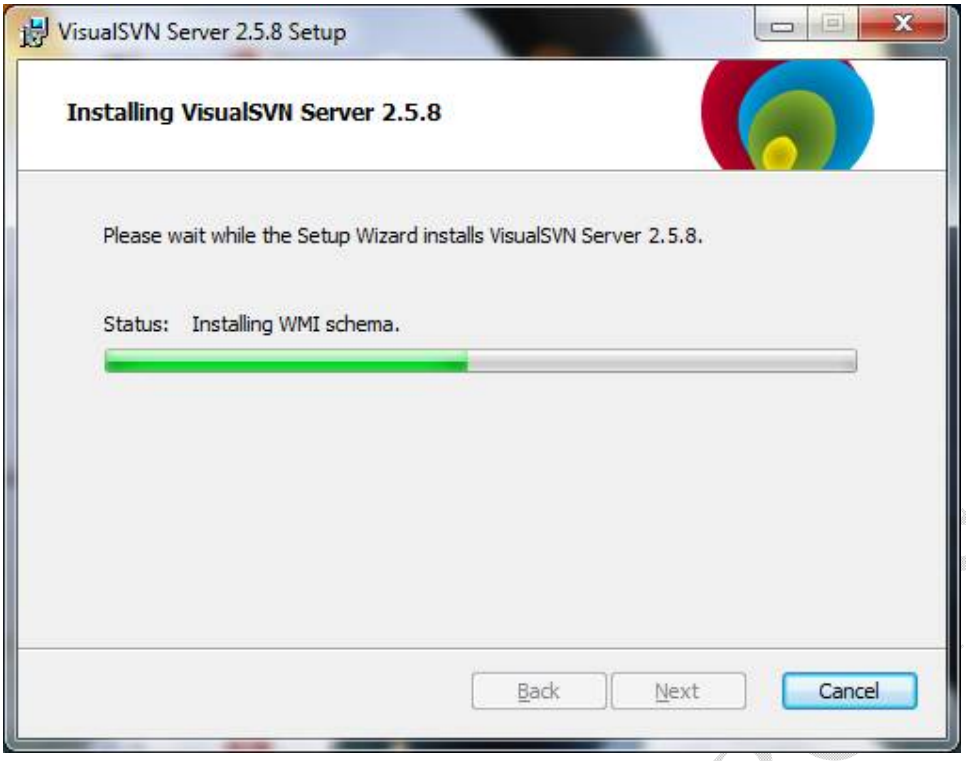
Click Next

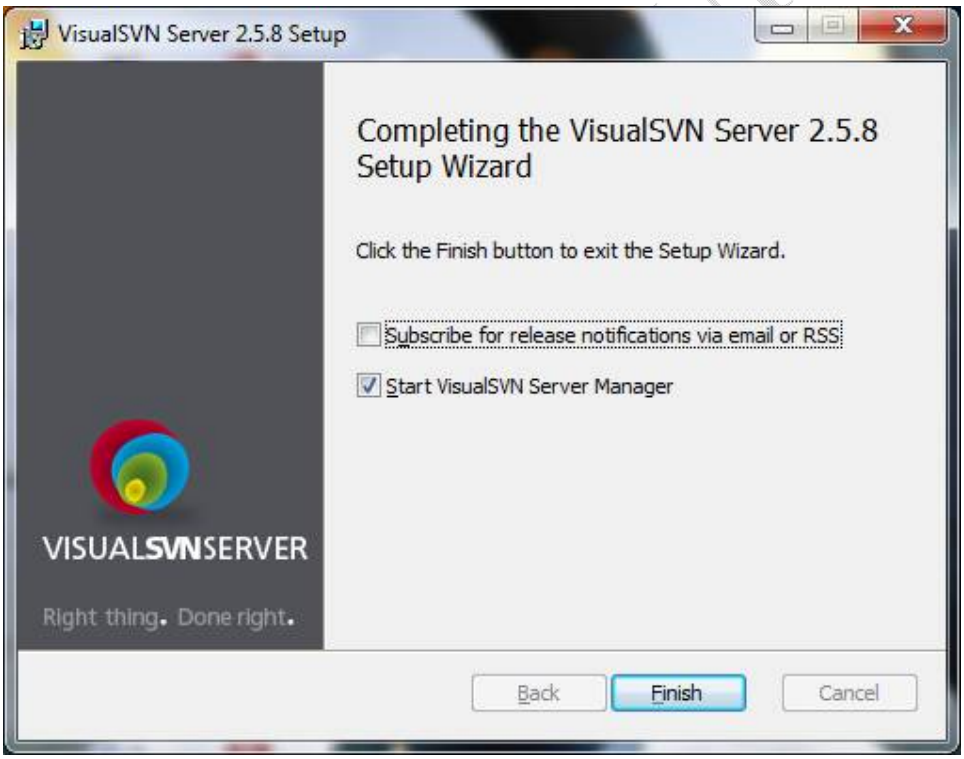


Click Next

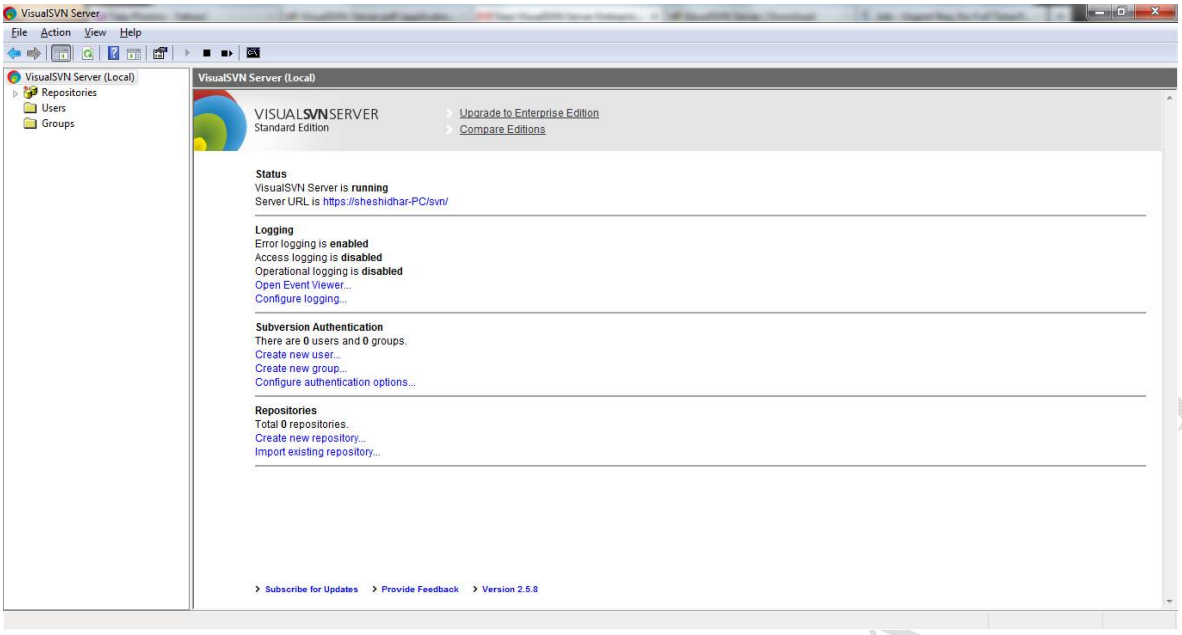


Click Install



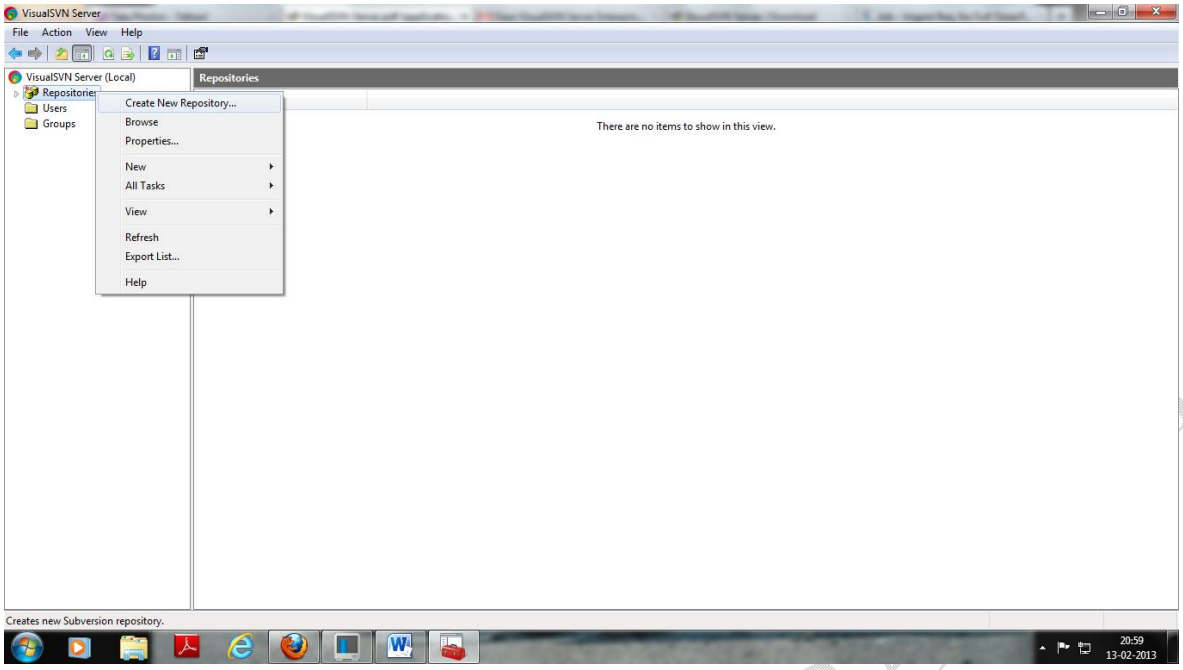


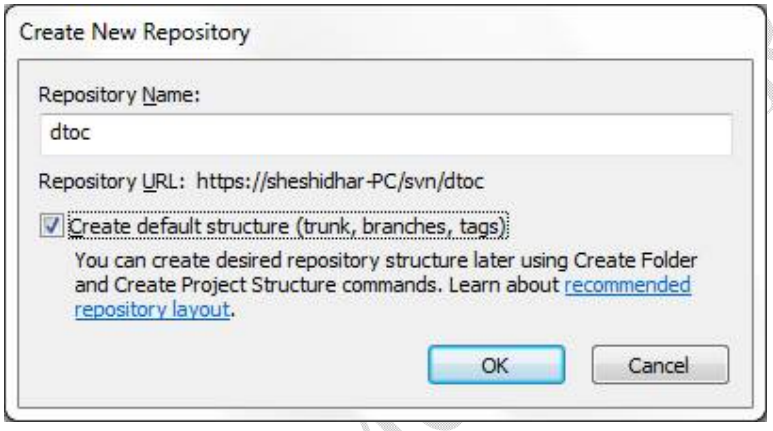
Click Finish



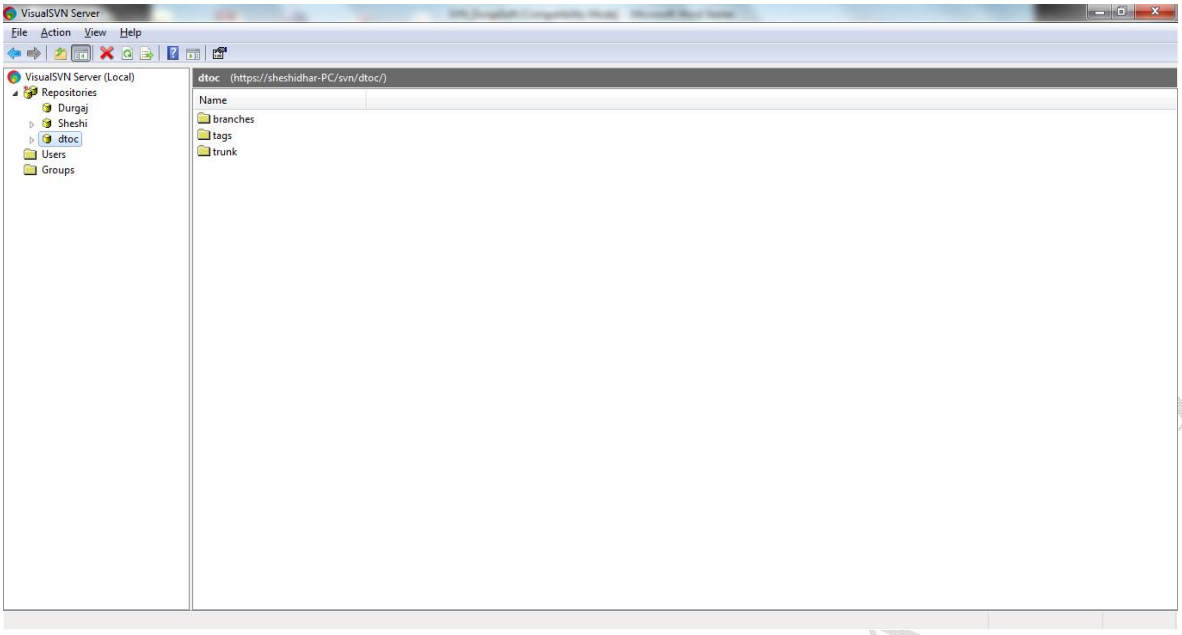
SVN Tortoise Server screen.

Steps to create Repository:

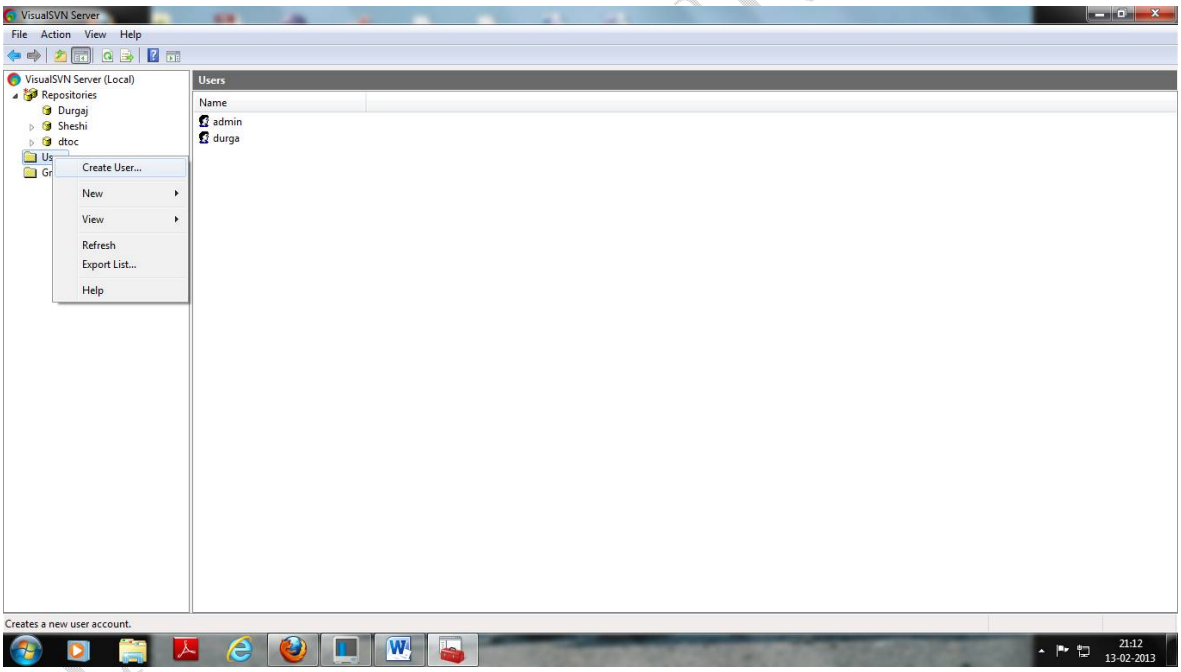

VisualSVN Server
File Action View Help
VisualSVN Server (Local)
Repositories
Users
Groups
Repositories
Create New Repository...
Browse
Properties...
New
All Tasks
View
Refresh
Export List...
Help
There are no items to show in this view.
Creates new Subversion repository.
20:59
13-02-2013

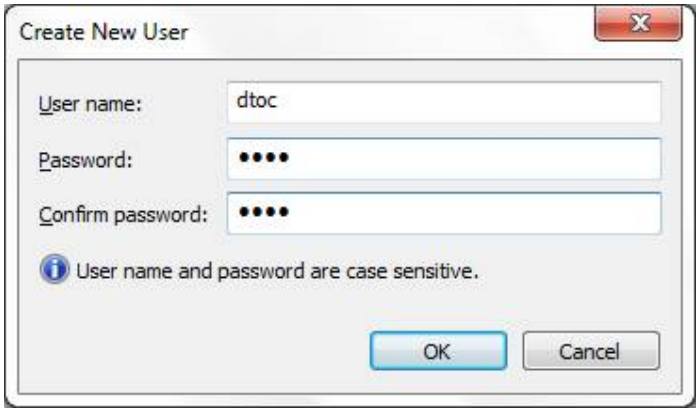

Create New Repository
Repository Name:
dtoc
Repository URL: https://sheshidhar-PC/svn/dtoc
Create default structure (trunk, branches, tags)
You can create desired repository structure later using Create Folder and Create Project Structure commands. Learn about recommended repository layout.
OK
Cancel

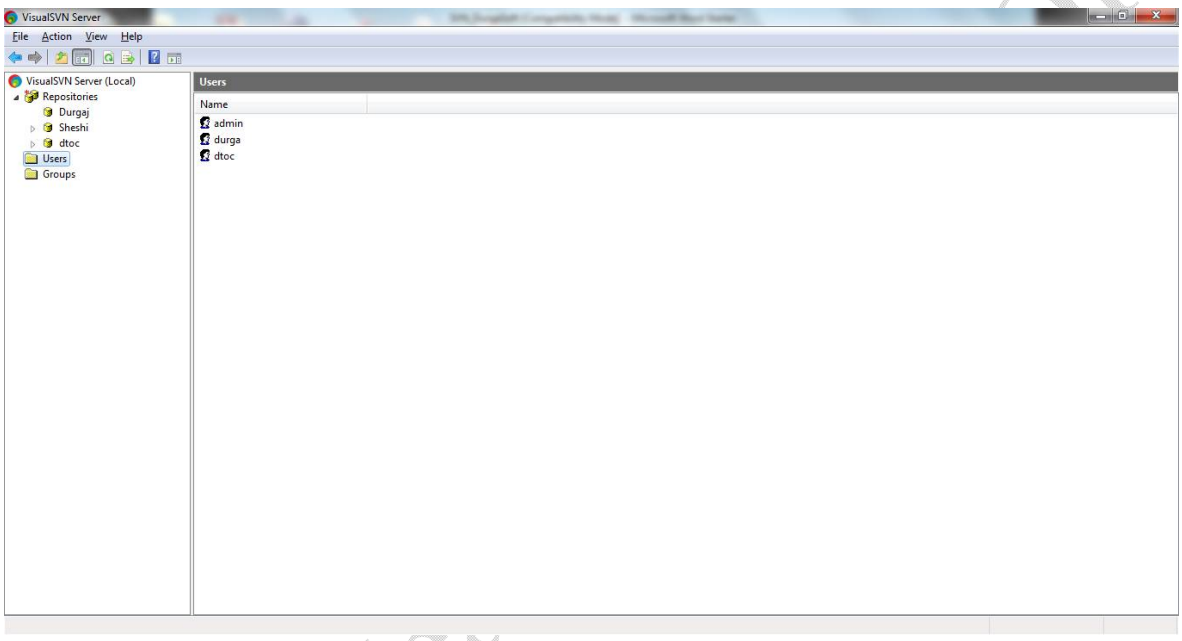

FREE TRAINING VIDEOS
YouTube
3000+ VIDEOS
www.youtube.com/durgasoftware



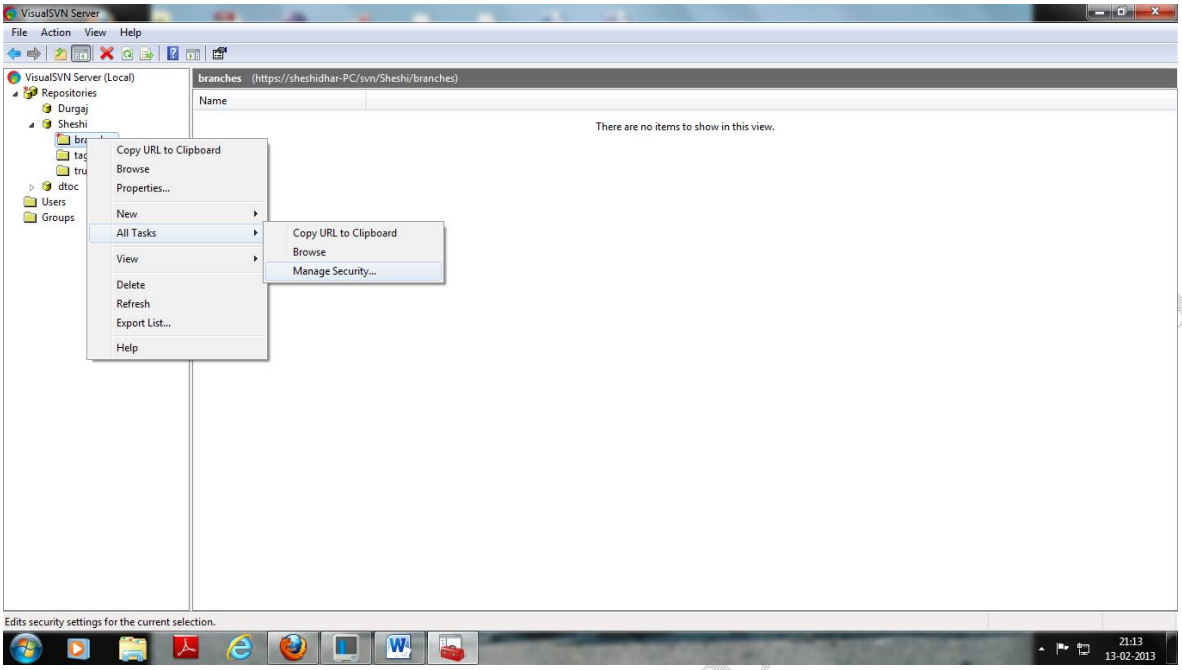
Steps to Create User:



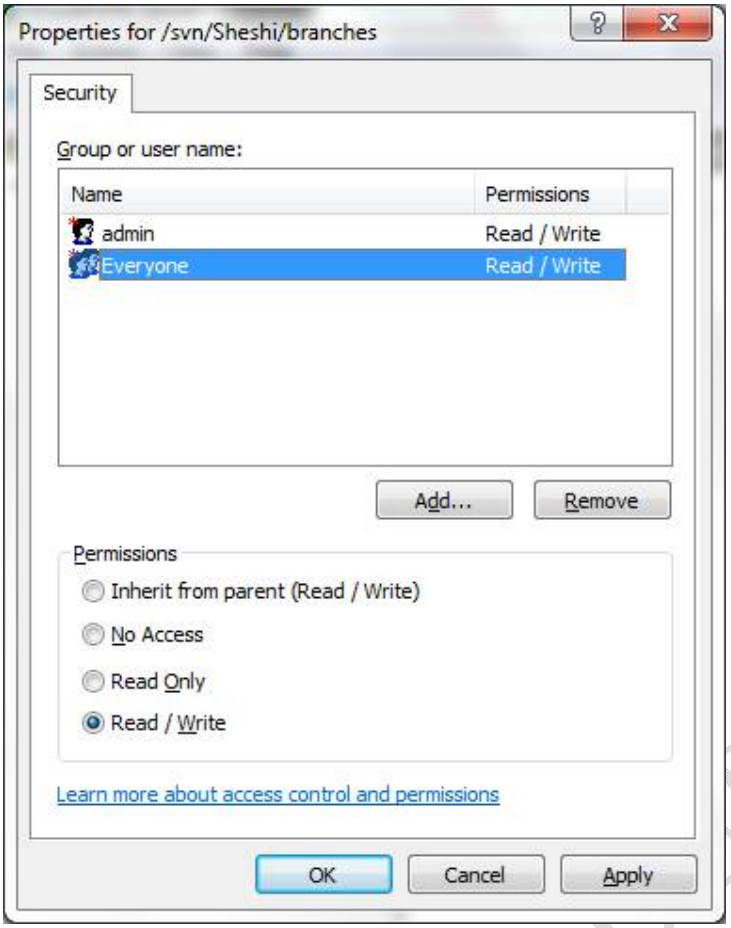

Create New User
User name:
dtoc
Password:
Confirm password:
User name and password are case sensitive.
OK
Cancel


VisualSVN Server
File Action View Help
VisualSVN Server (Local)
Repositories
Durgaj
Sheshi
dtoc
Users
Groups
Users
Name
admin
durga
dtoc

Steps: Adding User read/write permissions to access Branch/Trunk



Repository url: https://sheshidhar-pc/svn/Sheshi/branches/



SVN Plugin Configuration in Eclipse:

In Eclipse, select the menu "Help > Install New Software".

Enter the Update Site, depending on your version of subversion you installed:

If you installed Subversion 1.6: `http://subclipse.tigris.org/update_1.6.x`

If you installed Subversion 1.7: http://subclipse.tigris.org/update_1.8.x

Select the following items to be installed:

- Subclipse
- Subversion Client Adapter
- Subversion JavaHL Native Library Adapter
- SVNKit Client Adapter

After the installation, restart Eclipse.

To verify that the integration works, try the following steps:

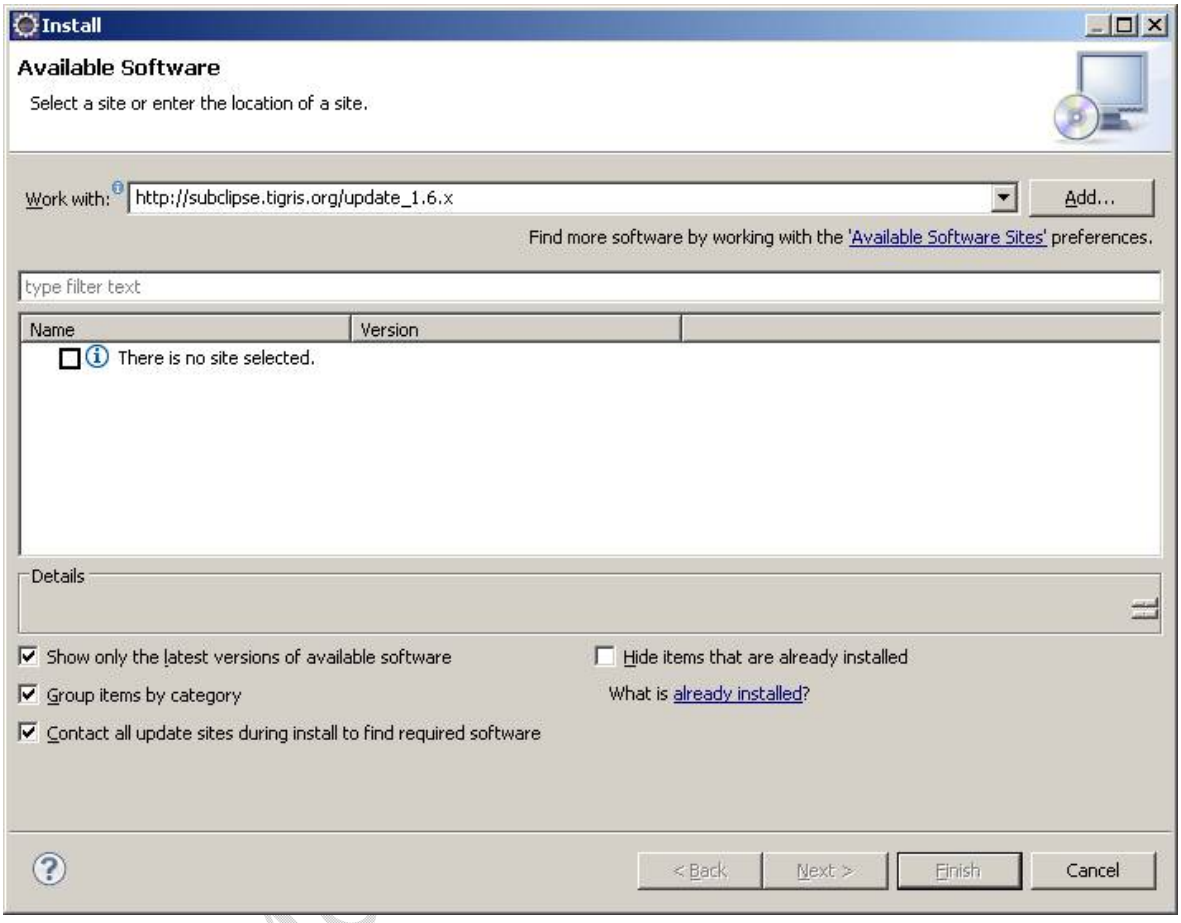
- Open the "SVN Repository Exploring" perspective (menu Window > Open Perspective > Other)

- Add the following SVN Repository Location:

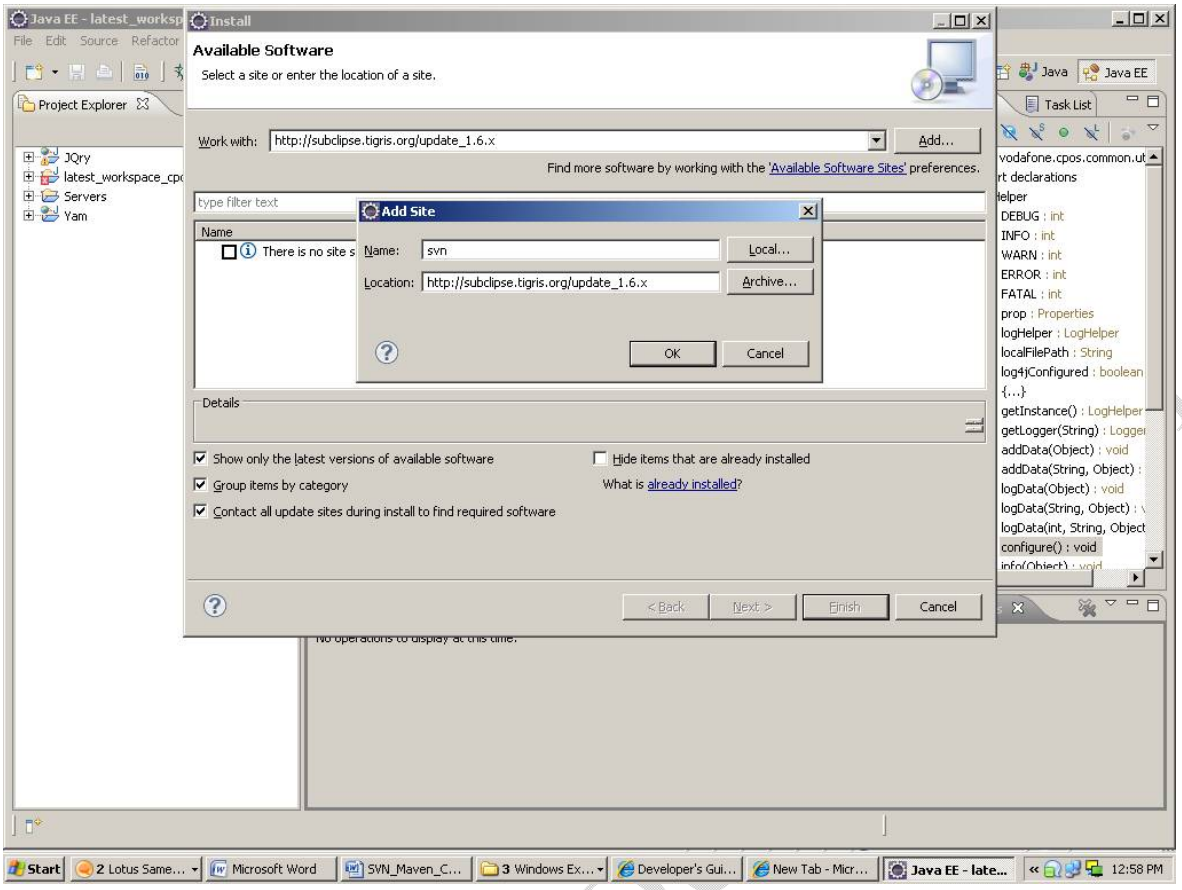
  `https://matsim.svn.sourceforge.net/svnroot/matsim`

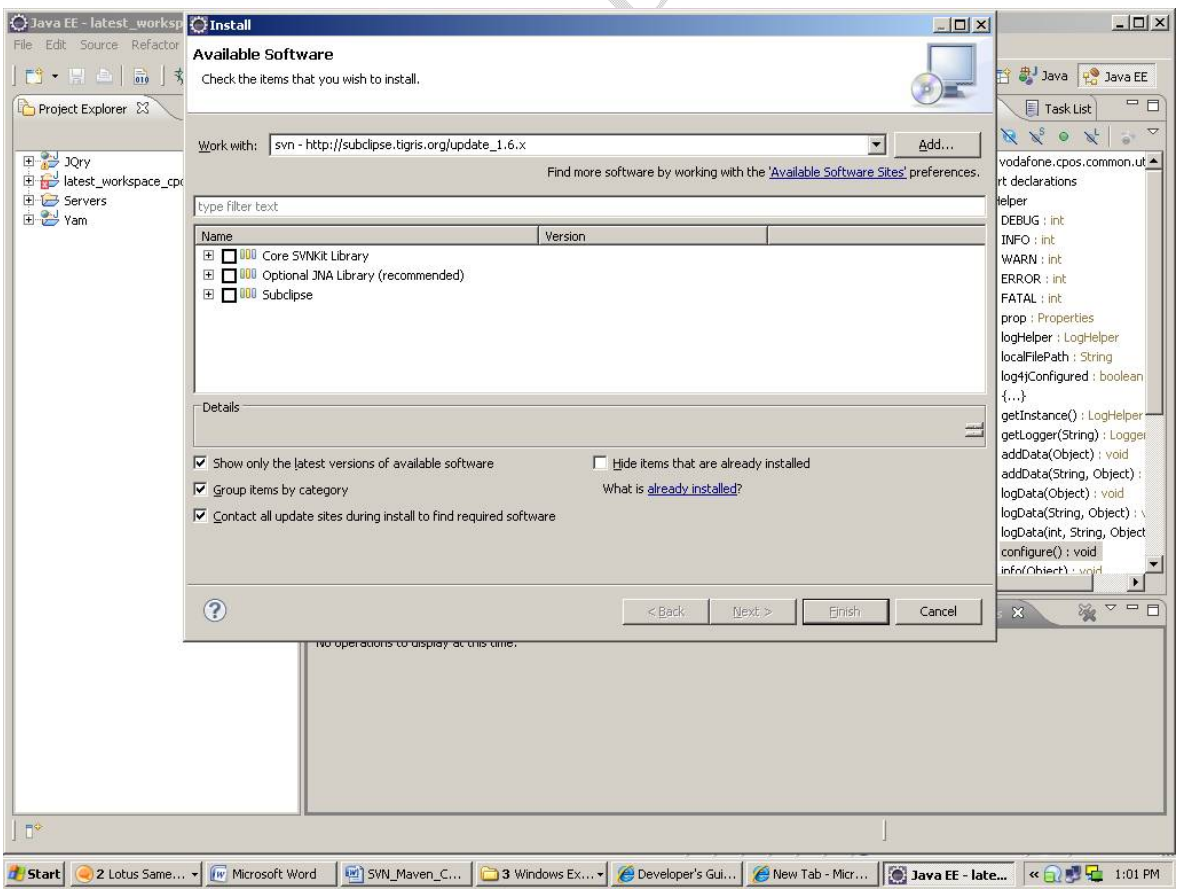
  To add it, click on the yellowish icon with "svn" and the plus sign.

- Test if you can browse to the sub-directories, e.g. `matsim/trunk`.

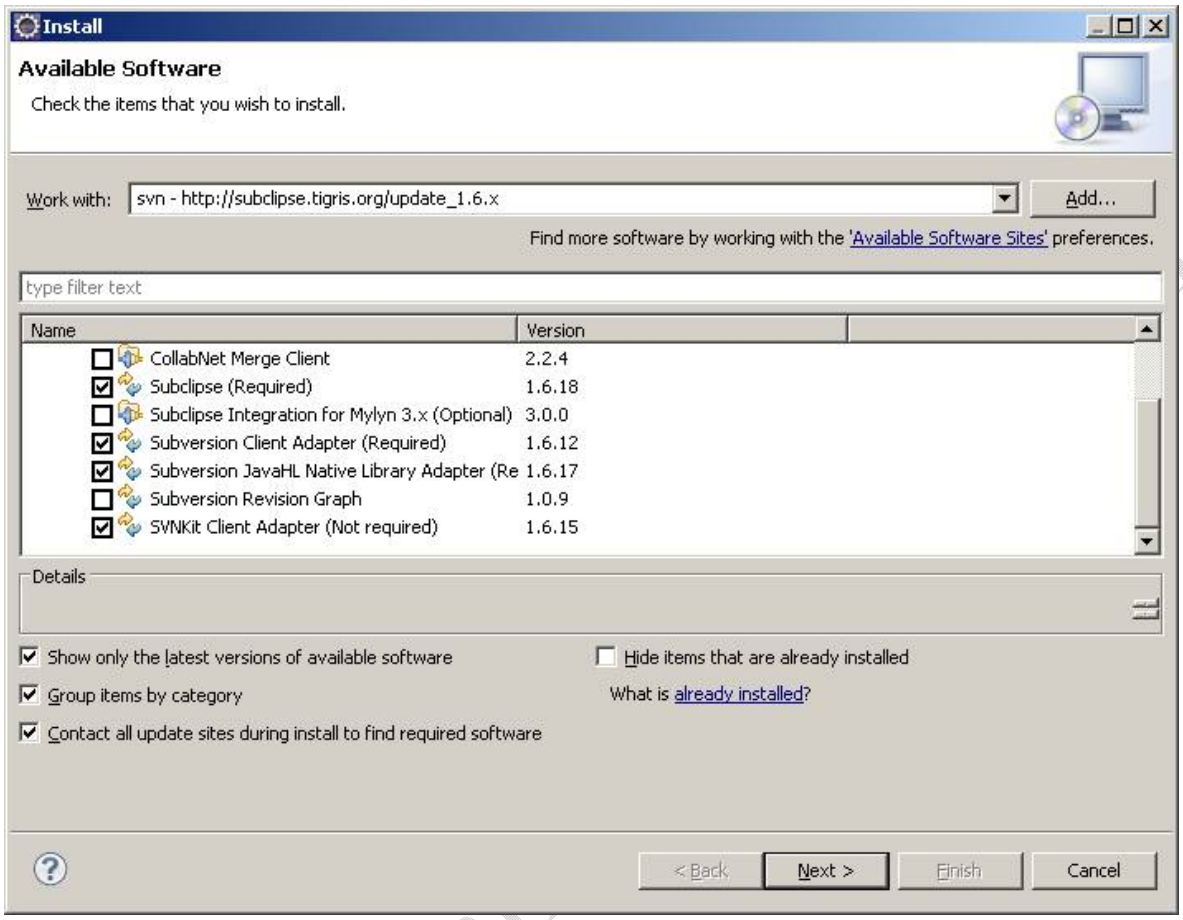


Click Add

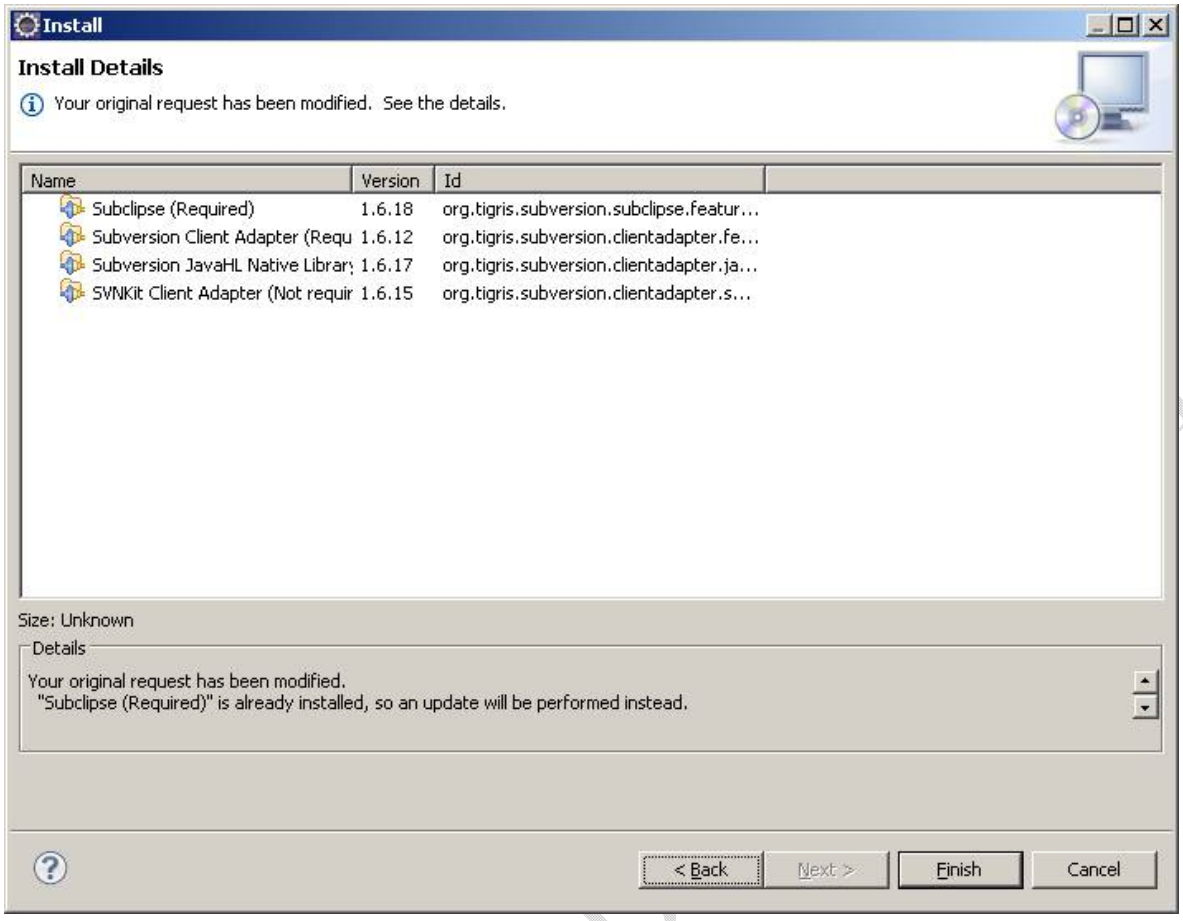

Install
Available Software
Select a site or enter the location of a site.
Work with: http://subclipse.tigris.org/update_1.6.x
Add...
Find more software by working with the 'Available Software Sites' preferences.
Add Site
Name: svn
Location: http://subclipse.tigris.org/update_1.6.x
Local...
Archive...
OK
Cancel
Details
Show only the latest versions of available software
Group items by category
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Finish
Cancel


Install
Available Software
Check the items that you wish to install.
Work with: svn - http://subclipse.tigris.org/update_1.6.x
Add...
Find more software by working with the 'Available Software Sites' preferences.
Name
Version
Core SVNKit Library
Optional JNA Library (recommended)
Subclipse
Details
Show only the latest versions of available software
Group items by category
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Finish
Cancel

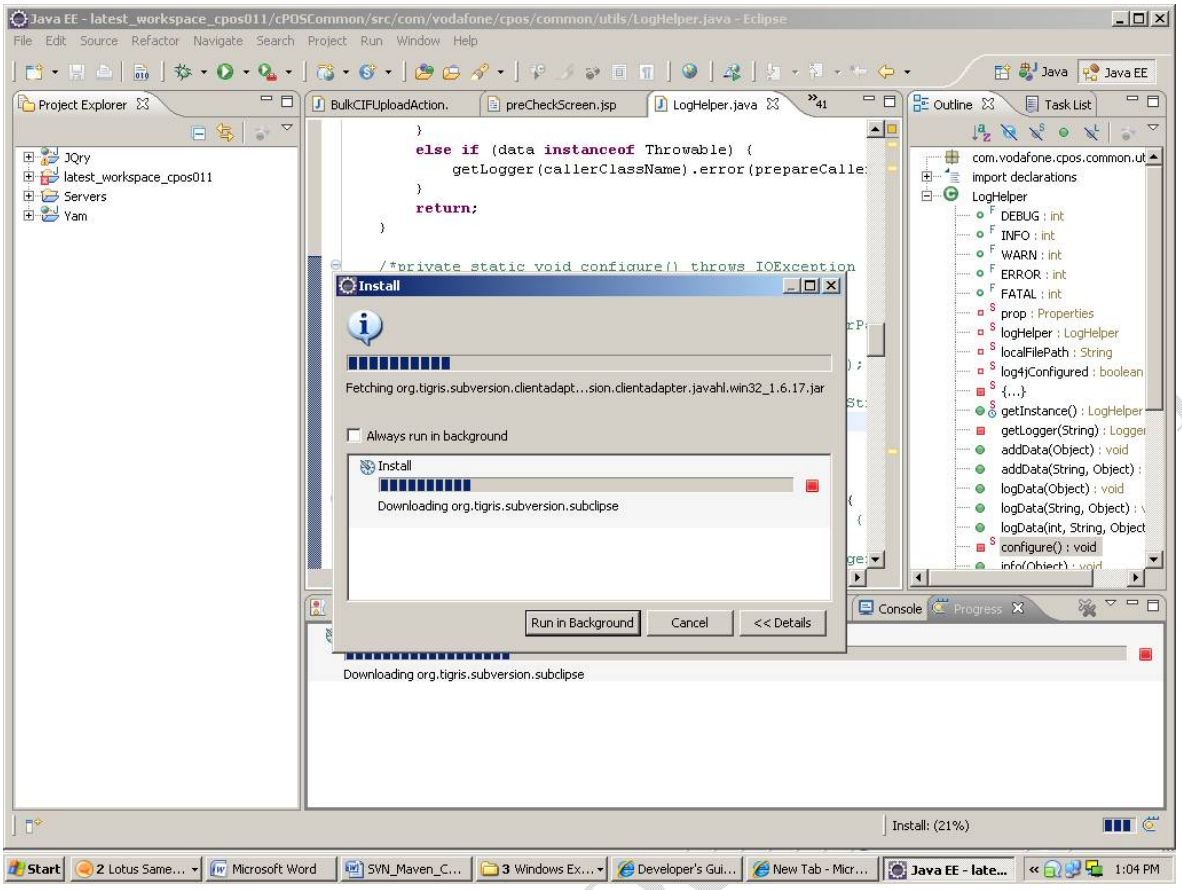
Expand Subclipse:



Click on next



Click Finish



After once installed it will prompts to restart the start system


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com