Software Engineering Tools Lab

Assignment No-6

PRN: 2019BTECS00090

Name: Vishal Chauhan

Batch: T8

Q 1. What is Microsoft's VSS? Provide the information of VSS tool with respect to below points.

a. Owner/ developer

b. Brief information/introduction

c. Basic operations involved

d. Advantages

e. Disadvantages

Ans:

a. Owner / Developer:

Originally developed by One Tree Software Company and later acquired by Microsoft.

b. Microsoft Visual SourceSafe is a file-level version control system that permits many types of organizations to work on several project versions at the same time. This capability is particularly beneficial in a software development environment, where it is used in maintaining parallel code versions. However, the product can also be used to maintain files for any other type of team.

At a minimum, Visual SourceSafe does the following:

- Helps protect your team from accidental file loss.
- Allows back-tracking to earlier versions of a file.
- Supports branching, sharing, merging, and management of file releases.
- Tracks versions of entire projects.
- Tracks modular code (one file that is reused, or shared, by multiple projects).

c. Some operations involved in Visual SourceSafe are:

Add:

Add is the starting point of all VSS file operations. Before you can do Get, Check In, Check Out or other operations, the file needs to be under the control of VSS first.

You can add individual files and folders into VSS database.

GET:

When you want to view/get a file or project, but not modify it, you can use the Get or View command. Get copies the file or project from the current project into your working folder.

Get Latest Version, which retrieves the most recent version of a file or a project to your working folder, is the most commonly used command in VSS.

Check Out:

To make changes to a file you must first check it out of the VSS database. When you Check Out an item, VSS retrieves the latest copy of the file to your working folder and make it writable.

There are exclusive check out and non-exclusive check out. If a file is checked out exclusively, it cannot be checked out by other users.

Please always check out a file first before making any changes for the following reasons:

1. If a file is not checked out, it cannot be checked in;
2. If a file is not checked out, your local copy may not be the latest copy. Not working on a latest copy may cause extra work, overwriting other's changes, confusion and many other problems;
3. If we change a file without checking it out first, other team members can check out the file and make changes to it, which can cause huge problem if parallel changes to a file is not desired.

Check In:

After you have finished the changes, you need to Check In the updated file back into VSS. When we say VSS keeps all your previous changes, we are not saying that VSS keeps every change you make, we are saying that VSS keeps all the versions you checked in. If you do not check in your file, there is no way that VSS knows that you have changed the file and it should keep it.

Q 2. Create a SVN repository and perform below operations on that repository using SVN. Also explain below operations. (First install Visual SVN server and Tortoise SVN client for the same)

a. Revert
b. Import
c. Checkout

d. Commit

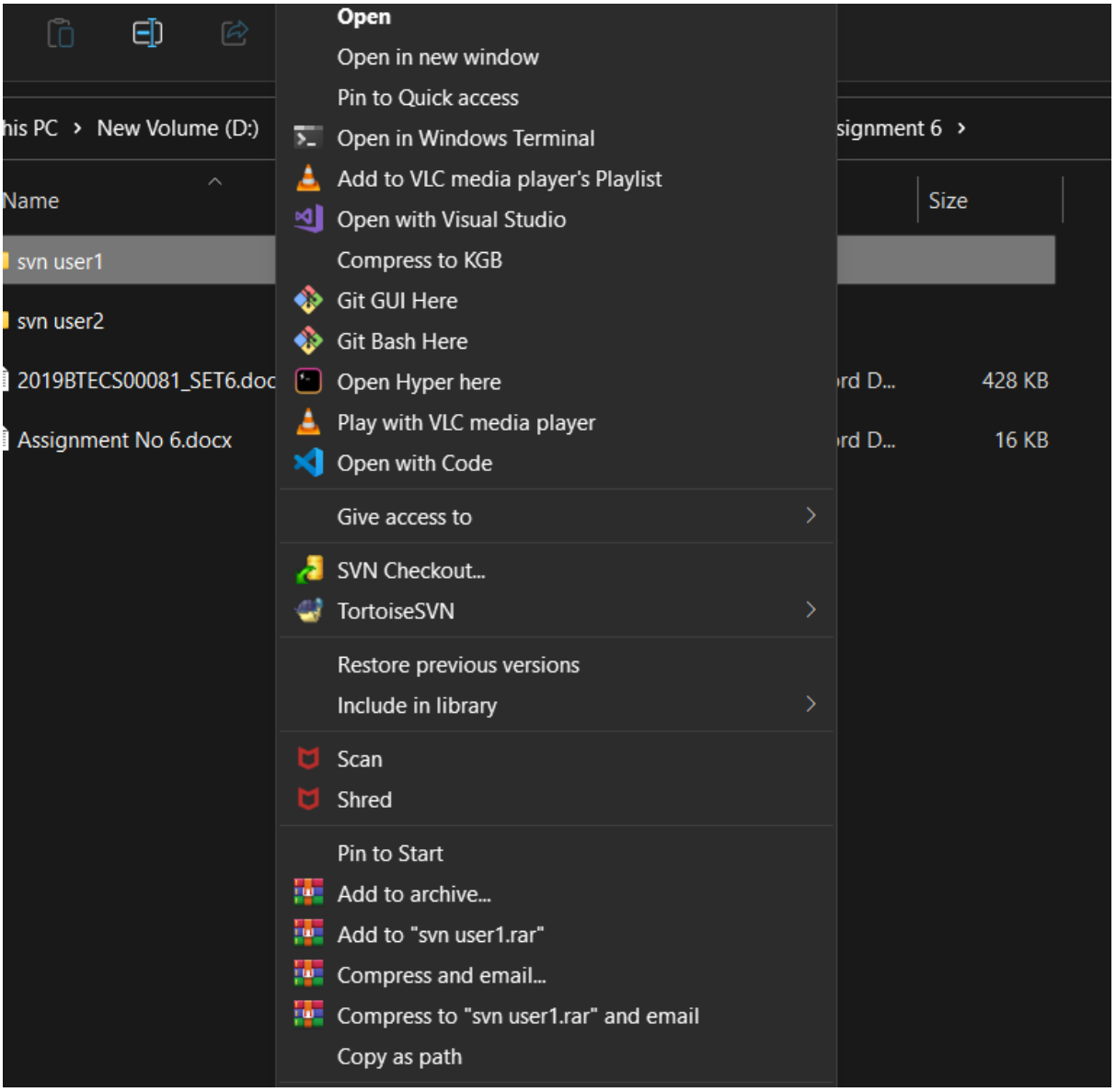
e. Update

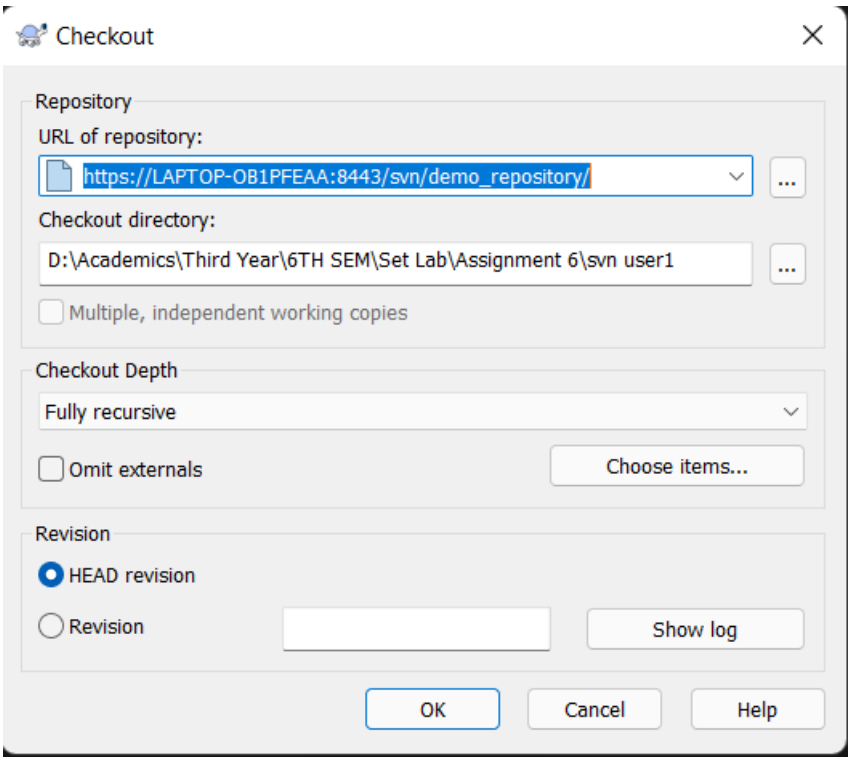
f. Copy

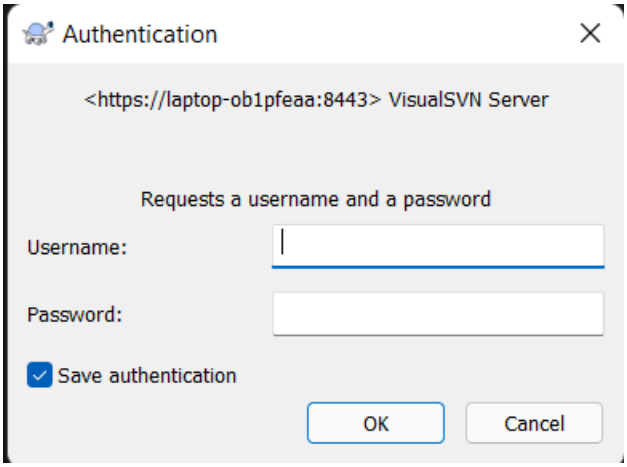
Ans:

Output:

Creating a SVN repository and user:

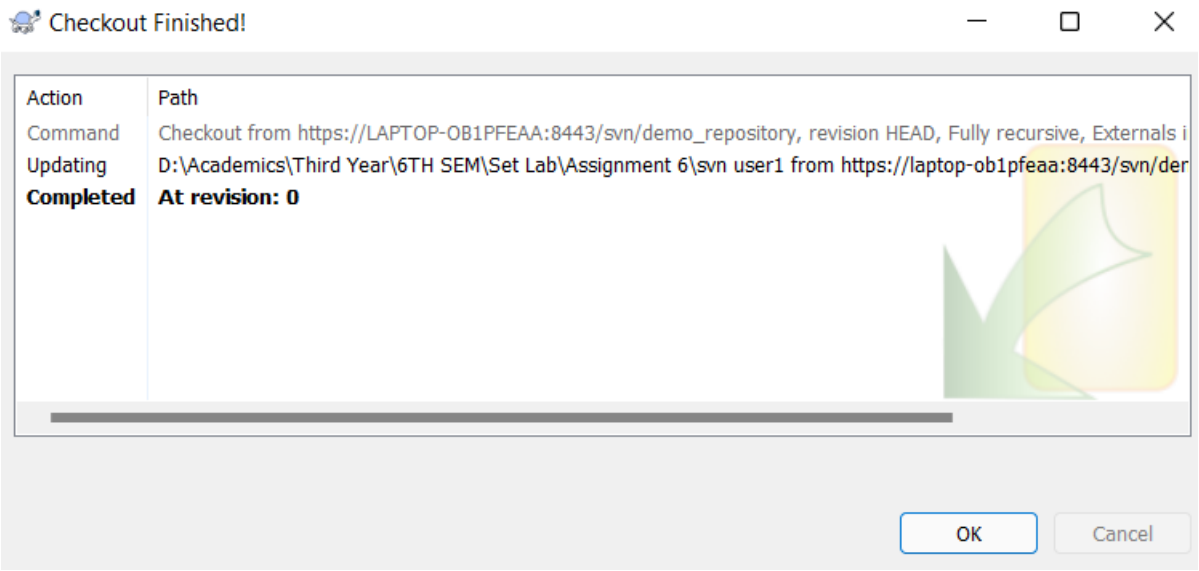






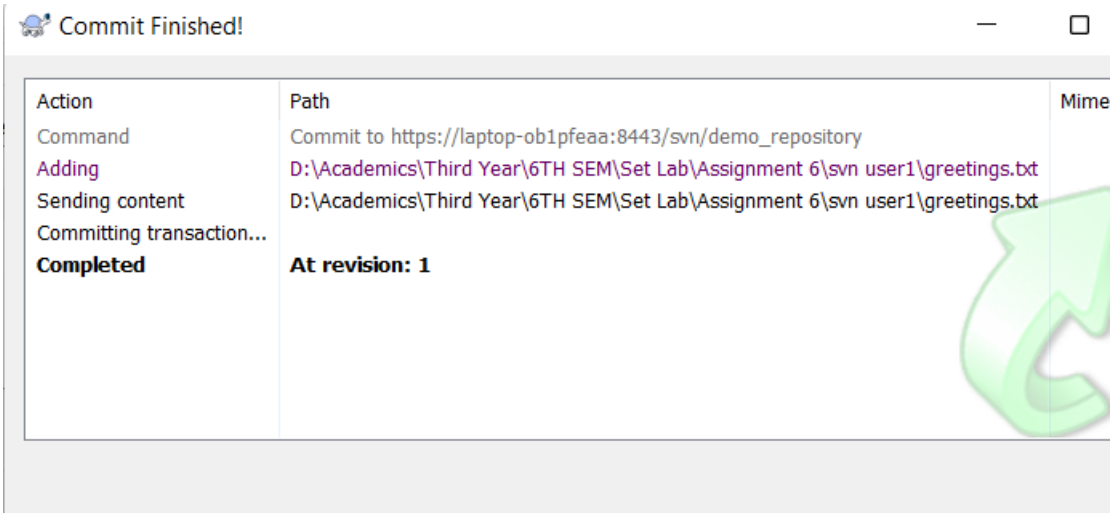
Add password for user Vishal

Checkout Completed



creating a file in SVN user1 folder

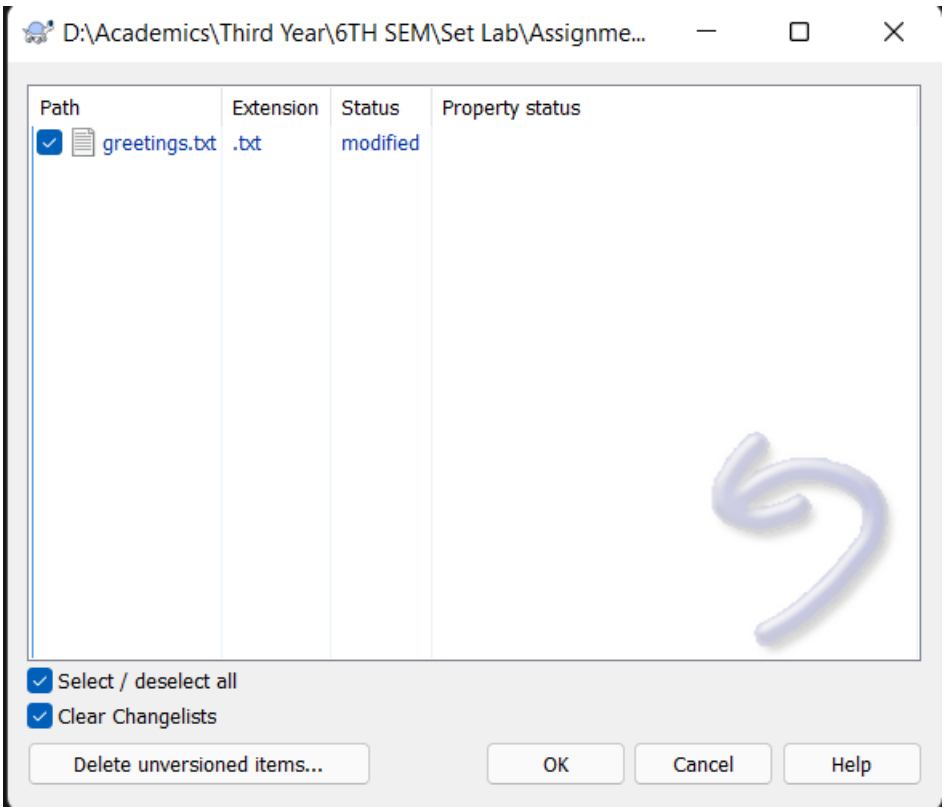
commit from Vishal

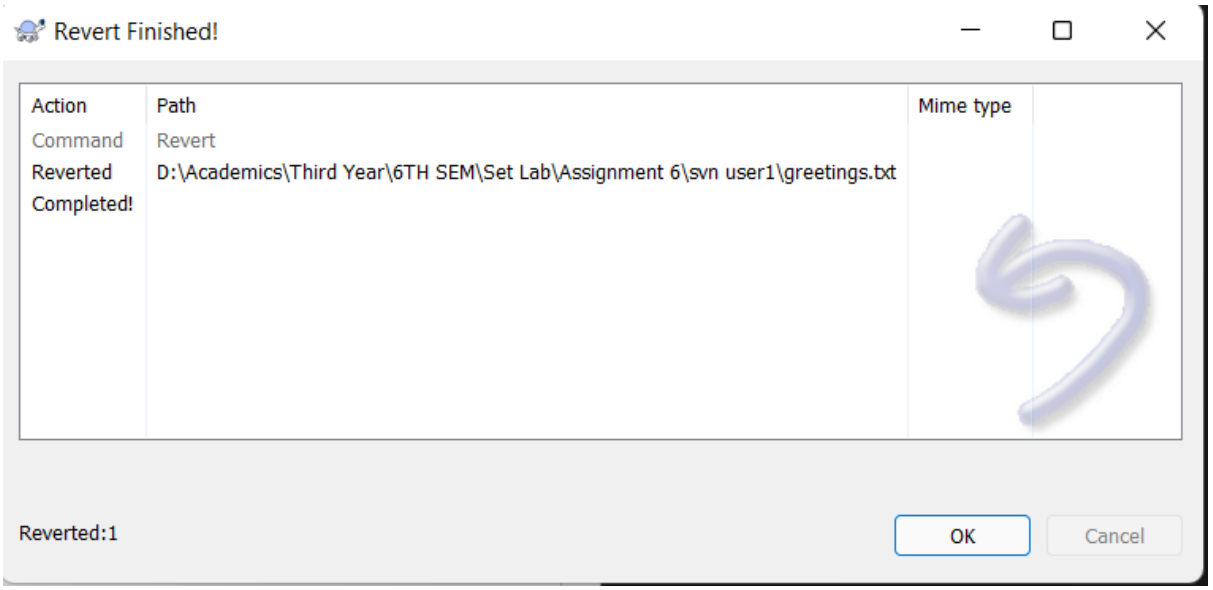


Updated changes

changes saved locally in greeting at SVN user1.

Reverting the local changes and updating it with actual content in repository





reverted changes are visible

Q 3. Perform below operations using CVS

a. cvs checkout

b. cvs update

c. cvs add

d. cvs remove

e. cvs commit

Ans:

Q 4. Differentiate Between The Git & SVN Repository?

Ans:

a. Directory Structure:

Each reference, or labeled snapshot of a commit, in a project is organized within specific subdirectories, such as trunk, branches, and tags. For example, an SVN project with two features under development might look like this:

```
sample_project/trunk/README.md
sample_project/trunk/lib/widget.rb
sample_project/branches/new_feature/README.md
sample_project/branches/new_feature/lib/widget.rb
sample_project/branches/another_new_feature/README.md
sample_project/branches/another_new_feature/lib/widget.rb
```

An SVN workflow looks like this:

- The trunk directory represents the latest stable release of a project.
- Active feature work is developed within subdirectories under branches.
- When a feature is finished, the feature directory is merged into trunk and removed.

Git projects are also stored within a single directory. However, Git obscures the details of its references by storing them in a special .git directory. For example, a Git project with two features under development might look like this:

```
sample_project/.git
sample_project/README.md
sample_project/lib/widget.rb
```

A Git workflow looks like this:

A Git repository stores the full history of all of its branches and tags within the .git directory.

- The latest stable release is contained within the default branch.
- Active feature work is developed in separate branches.
- When a feature is finished, the feature branch is merged into the default branch and deleted.

Unlike SVN, with Git the directory structure remains the same, but the contents of the files change based on your branch.

Including SubProject:

A subproject is a project that's developed and managed somewhere outside of your main project. You typically import a subproject to add some functionality to your project without needing to maintain the code yourself. Whenever the subproject is updated, you can synchronize it with your project to ensure that everything is up-to-date.

In SVN, a subproject is called an SVN external. In Git, it's called a Git submodule. Although conceptually similar, Git submodules are not kept up-to-date automatically; you must explicitly ask for a new version to be brought into your project.

Preserving History:

SVN is configured to assume that the history of a project never changes. Git allows you to modify previous commits and changes using tools like git rebase.

Q 5. What is “branch”, “tag” And “trunk” In SVN?

Ans:

Trunk would be the main body of development, originating from the start of the project until the present.

Branch will be a copy of code derived from a certain point in the trunk that is used for applying major changes to the code while preserving the integrity of the code in the trunk. If the major changes work according to plan, they are usually merged back into the trunk.

Tag will be a point in time on the trunk or a branch that you wish to preserve. The two main reasons for preservation would be that either this is a major release of the software, whether alpha, beta, RC or RTM, or this is the most stable point of the software before major revisions on the trunk were applied.

In open source projects, major branches that are not accepted into the trunk by the project stakeholders can become the bases for forks -- e.g., totally separate projects that share a common origin with other source code.

The branch and tag subtrees are distinguished from the trunk in the following ways:

Subversion allows sysadmins to create hook scripts which are triggered for execution when certain events occur; for instance, committing a change to the repository. It is very common for a typical Subversion repository implementation to treat any path containing "/tag/" to be write-protected after creation; the net result is that tags, once created, are immutable (at least to "ordinary" users). This is done via the hook scripts, which enforce the immutability by preventing further changes if tag is a parent node of the changed object.

Subversion also has added features, since version 1.5, relating to "branch merge tracking" so that changes committed to a branch can be merged back into the trunk with support for incremental, "smart" merging.

Q6 . How CVS is different from SVN?

CVS and SVN are two popular version control systems that are used to manage changes to software code and other digital assets. While both systems are designed to help teams collaborate on projects, they differ in several key ways.

CVS is a centralized version control system, which means that all changes to a codebase are managed through a single server. This can make branching and merging more tedious and error-prone, as well as limit the speed and efficiency of large repositories. CVS also uses a less sophisticated file format, which can make renaming and deleting files more cumbersome. However, CVS does offer extensive support for file locking, which can be helpful in certain scenarios.

In contrast, SVN offers both centralized and decentralized options, which allow teams to choose the approach that best fits their needs. SVN also supports atomic commits, which ensure that changes are committed to the codebase as a single, cohesive unit. This can make merging and conflict resolution much easier and more robust. Additionally, SVN has a more sophisticated authentication and security system, which can help protect sensitive code and data.

Overall, while CVS may still be preferred in some cases, SVN is generally considered to be a more modern and capable version control system. Its support for atomic commits, easier branching and merging, and more sophisticated conflict resolution make it a popular choice for many development teams.

Feature	CVS	SVN
Type of system	Centralized	Centralized or Decentralized (with Git-like SVN)
Branching and merging	Tedious and error-prone	Easier and more robust
Renaming and deleting	Cumbersome	Easy
Atomic commits	Not supported	Supported
File locking	Used extensively	Optional
Repository storage	Uses RCS file format	Uses a custom database format
Performance	Slower for large repositories	Faster for large repositories
Conflict resolution	Basic, limited functionality	More sophisticated
Authentication/Security	Basic	More advanced, including SSL and SSH