**Software Engineering Tools Lab**

**Assignment No-5**

**Module 4 & 5- Understanding version control using VSS and Managing code using SVN**

**PRN: 2020BTECS00079**

**Name: Saurabh Khadsang**

---

**1) What is version control system and why it is important?**

Version control system (VCS) is a software tool that helps manage changes to files and projects over time, allowing multiple people to collaborate on the same project without overwriting each other's work. It provides a record of all changes made to files, and enables users to track and compare different versions of those files.

VCS is important because it offers several benefits, including:

1. Collaboration: VCS enables multiple developers to work on the same project, without interfering with each other's work. This is because the system keeps track of all changes made to files, and allows developers to merge their changes together seamlessly.
2. Backup and Restore: With VCS, developers can keep multiple versions of files, which can be easily restored in the event of data loss, corruption or other issues.
3. Traceability: VCS makes it easy to track changes to files and identify who made those changes, when they were made, and why. This can be important for compliance, audit and troubleshooting purposes.
4. Experimentation: VCS allows developers to create and experiment with different versions of their code, without affecting the main codebase. This can help them to test new features or try out different approaches to a problem.

**2) Illustrate different types of version control system with example.**

There are several types of version control systems (VCS), each with its own unique approach to tracking and managing changes to files and projects. Here are some examples of the most common types of VCS:

1. Centralized Version Control System (CVCS): CVCS is a type of version control system where all changes to files are managed in a central repository. Developers must check out files from the central repository, make their changes, and then check them back in. Examples of CVCS are Concurrent Versions System (CVS), Perforce, and Subversion (SVN)
2. Distributed Version Control System (DVCS): DVCS is a type of version control system where developers have a complete copy of the code repository on their local machine. Changes are made locally, and then pushed or pulled to a central repository. Examples of DVCS include Git.

3. File-based/ Local Version Control System: File-based version control systems are a type of VCS that track changes at the file level. They are often used for simple projects, such as personal coding projects or small teams.
4. Cloud-based Version Control System: Cloud-based version control systems are a type of VCS that are hosted in the cloud, enabling developers to access their code from anywhere with an internet connection. Examples of cloud-based VCS include GitHub, GitLab, and Bitbucket.

**3) Perform below operations using CVS**

**a. CVS checkout**

```
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5> cvs -d "F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\CVS" checkout -
d workspace workspace
cvs checkout: Updating workspace
```

**b. CVS update**

```
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5> cd workspace
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs add test.txt
cvs add: nothing known about test.txt
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs add test.txt
cvs add: scheduling file `test.txt' for addition
cvs add: use 'cvs commit' to add this file permanently
```

**c. CVS adds**

```
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5> cd workspace
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs add test.txt
cvs add: nothing known about test.txt
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs add test.txt
cvs add: scheduling file `test.txt' for addition
cvs add: use 'cvs commit' to add this file permanently
```

**d. CVS removes**

```
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs remove .\test.txt
cvs remove: file `test.txt' still in working directory
cvs remove: 1 file exists; remove it first
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs commit -m "Removed tests file "
cvs commit: Examining .
```

**e. CVS commit**

```
PS F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\workspace> cvs commit -m "Added new file"
cvs commit: Examining .
RCS file: F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\CVS/workspace/test.txt,v
done
Checking in test.txt;
F:\z.DOdo\WCE Assignment\Sem 6\SAT\5\CVS/workspace/test.txt,v  <--  test.txt
initial revision: 1.1
done
```

**4) Differentiate Between the Git & SVN Repository?**

Both Git and SVN are version control systems that help manage source code changes over time. However, there are some differences between them:

Distributed vs Centralized: Git is a distributed version control system, which means that each user has a full copy of the repository on their local machine. This allows for easier branching, merging, and offline work. SVN, on the other hand, is a centralized version control system, where the repository is stored on a central server and users checkout a working copy of the code.

Speed: Git is known to be faster than SVN, especially for operations like branching and merging. This is because Git stores the entire repository locally, while SVN requires communication with the central server for many operations.

Branching and Merging: Git's branching and merging capabilities are considered more powerful and flexible than SVN's. In Git, creating and merging branches is a very common and easy operation, and Git's merging algorithms are very good at handling complex merge scenarios. In SVN, branching and merging can be more complex and time-consuming, especially for larger projects.

Workflow: Git's workflow is often more suitable for modern development practices like Agile and DevOps, where frequent code changes and deployments are common. Git's branching and merging capabilities, along with its distributed nature, make it easier to manage complex development workflows. SVN's centralized nature can make it more challenging to manage complex workflows.

Learning Curve: Git has a steeper learning curve than SVN, especially for developers who are new to version control. Git has many powerful features and a more complex command-line interface, which can make it harder for beginners to get started. SVN, on the other hand, has a simpler interface and fewer features, which can make it easier for beginners to learn. However, both Git and SVN have user-friendly graphical interfaces that can help simplify the learning process.

Overall, the choice between Git and SVN depends on the specific needs and requirements of the project and development team. For smaller projects or teams with less complex workflows, SVN may be a better choice. However, for larger projects or teams that require more advanced branching and merging capabilities, Git may be a better fit.

**5) What is “branch”, “tag” And “trunk” In SVN?**

In SVN (Subversion), "branch", "tag", and "trunk" are commonly used terms to refer to different paths within a repository:

1. Trunk: The "trunk" is the main line of development in a SVN repository. It represents the latest version of the code and is where active development takes place. All changes to the code are committed to the trunk by developers.

2. Branch: A "branch" is a copy of the codebase at a specific point in time. Developers create branches to work on a new feature or bug fix without affecting the main trunk. Changes made in a branch can be merged back into the trunk at a later time.
3. Tag: A "tag" is a snapshot of the codebase at a specific point in time that is labeled with a version number or release number. Tags are typically used to mark major releases or milestones in the development process. Unlike branches, tags are read-only and are not intended for further development.

In summary, trunk is the main line of development, branch is a copy of the codebase used for independent development, and tag is a labeled snapshot of the codebase used for marking releases or milestones.

**6) How CVS is different from SVN?**

CVS (Concurrent Versions System) and SVN (Subversion) are both version control systems used to manage source code and track changes made by developers. Here are some key differences between the two:

1. Architecture: CVS uses a client-server architecture, where a central server stores the repository and clients check out files from the server. SVN uses a client-server architecture as well, but with a more modern approach. SVN uses a single repository that contains the complete history of the project, while CVS stores each file's version history separately.
2. Handling binary files: CVS has limited support for handling binary files. Binary files are files that are not in text format and include images, audio files, and other non-text files. SVN handles binary files more efficiently, making it a better choice for projects that require a lot of binary files.
3. Renaming files and directories: Renaming files and directories in CVS can be difficult and can cause issues. SVN handles file and directory renaming more efficiently.
4. Branching and Merging: Branching and merging in CVS can be complex and difficult to manage, while SVN has improved branching and merging capabilities. SVN provides better support for merging changes from one branch to another.
5. Repository backup and recovery: In CVS, backing up and restoring repositories can be time-consuming and difficult. SVN provides better support for repository backup and recovery.

Overall, SVN is considered to be more advanced and modern than CVS, with better support for binary files, branching, merging, renaming, and repository backup and recovery. SVN has become the standard choice for version control in many software development projects, while CVS is used less frequently today.

**7) Demonstrate a display the app version in angular.**

To display the app version in an Angular application, you can follow these steps:

1. Create a new file named version.ts in the src/app directory.
2. In version.ts, define a constant variable that contains the version number of your application.

3. In your app.component.ts file, import the VERSION variable from the version.ts file.
4. Add a public property to the AppComponent class that references the VERSION constant.
5. In your app.component.html file, display the version number using interpolation.
6. Save your changes and run your application. The version number should now be displayed in your app.

Version.ts

```
export const VERSION = '1.0.0';
```

app.component.ts

```
export class AppComponent {
    public version = VERSION;
  }
```

**8) Build a simple web app with Express and Angular. Referance: https://www.geeksforgeeks.org/build-a-simple-web-app-with-expressangular/?ref=rp**

To build a simple web app with Express and Angular, follow these steps:

1. Install Node.js and npm (Node Package Manager) on your computer if you haven't already.
2. Open a command prompt or terminal window.
3. Create a new directory for your project and navigate to it.
4. Initialize a new Node.js project by running the command **npm init**.
5. Install Express by running the command **npm install express**.
6. Install Angular CLI by running the command **npm install -g @angular/cli**.
7. Use Angular CLI to create a new Angular project by running the command **ng new client**. This will create a new directory called **client** containing your Angular app.
8. Change into the **client** directory by running the command **cd client**.
9. Start the Angular app by running the command **ng serve**.
10. In a separate terminal window, navigate to the root directory of your project.
11. Create a new file called **index.js** and add the following code to it:

```
const express = require('express');
const app = express();

// handling CORS
app.use((req, res, next) => {
```

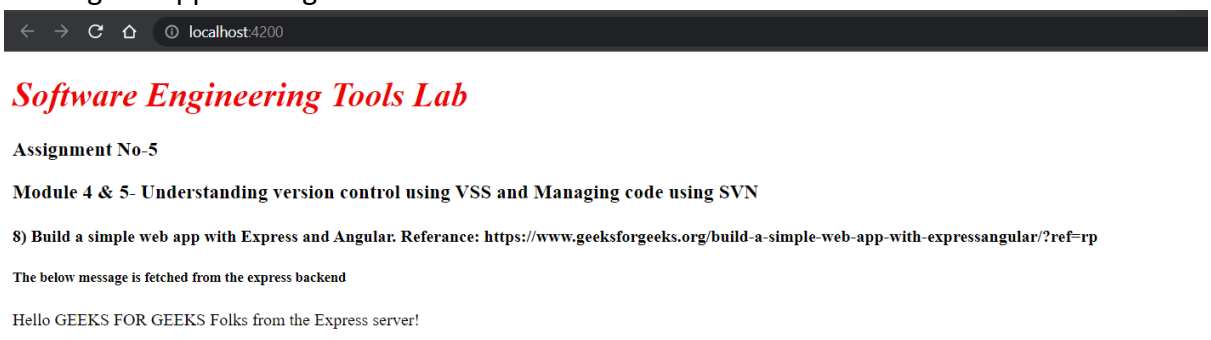
```
    res.header("Access-Control-Allow-Origin",
            "http://localhost:4200");
    res.header("Access-Control-Allow-Headers",
            "Origin, X-Requested-With, Content-Type, Accept");
    next();
});

// route for handling requests from the Angular client
app.get('/api/message', (req, res) => {
    res.json({ message:
            'server responding !' });
});

app.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

12. Run the command **npm install --save-dev nodemon** to install the nodemon package as a development dependency.
13. Update the **scripts** section of your **package.json** file to include the following line:

jsonCopy code

"start": "nodemon index.js"

14. Run the command **npm start** to start the server.
15. Open your web browser and go to **http://localhost:3000**. You should see your Angular app running.



**9) What is git version control?**

Git is a distributed version control system that allows you to track changes to files and collaborate with others on software development projects.

Version control systems (VCS) are tools used to manage changes to source code over time. They allow developers to collaborate on code and keep track of changes, enabling them to work on the same codebase without overwriting each other's work.

Git works by keeping track of changes to files in a repository. It creates a history of changes that can be accessed at any time, and it allows you to branch off from the main codebase to work on new features or bug fixes without affecting the main codebase. Once you've made changes, you can commit them to the repository, and Git will keep track of who made the changes, what changes were made, and when they were made.

Git also supports collaboration by allowing developers to share their changes with others through remote repositories, such as those hosted on GitHub, GitLab, or Bitbucket. This allows developers to work together on the same codebase and keep track of changes made by others.

## 10) Demonstrate creation of repository in git.

To create a new repository in Git, follow these steps:

1. Open a command prompt or terminal window.
2. Navigate to the directory where you want to create the repository.
3. Type the command **git init** and press Enter. This will create a new Git repository in the current directory.
4. If you want to add files to the repository, create them in the current directory or copy them to the directory using the appropriate commands.
5. Type the command **git add <file>** to add a file to the repository. You can use wildcards (*****) to add multiple files at once, or use **git add .** to add all files in the directory.
6. Once you have added all the files you want to include in the repository, type the command **git commit -m "Initial commit"** to commit the changes. Replace "Initial commit" with a brief description of the changes you made.
7. If you want to push the repository to a remote server (such as GitHub), create a new repository on the server and follow the instructions provided by the server to push the repository.