Software Engineering Assignment- 03
Version Control Tools

Megha C Savalgi(15IT220)

Introduction:

- Version control systems are a category of software tools that help a software team manage changes to source code over time.
- Version control software keeps track of every modification to the code in a special kind of database.
- If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.
- For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort.
- Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

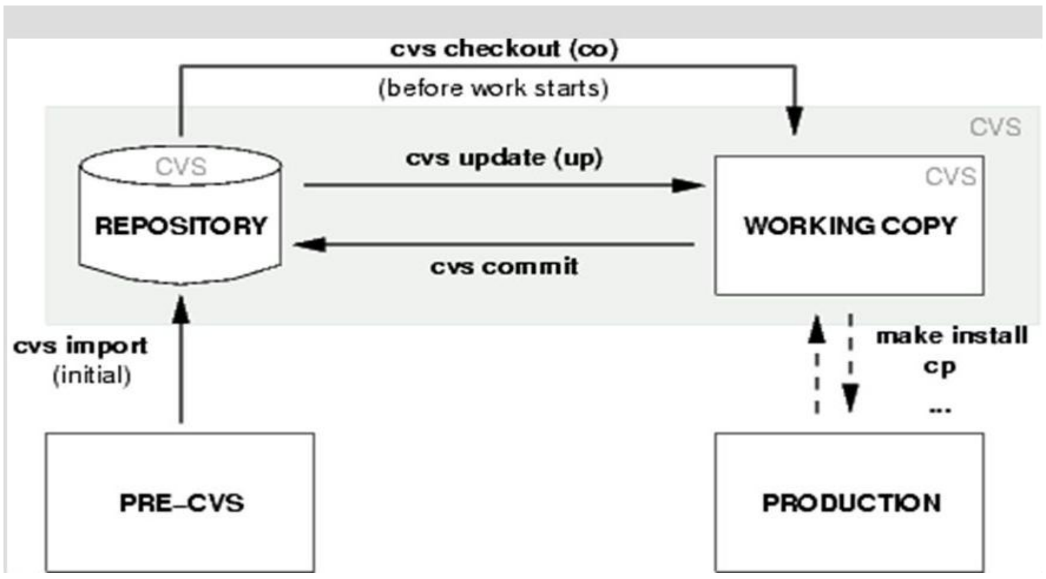
Software developers working in teams are continually writing new source code and changing existing source code. The code for a project is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

1. Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.
2. Changes made in one part of the software can be incompatible with those made by another developer working at the same time.
3. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

Centralized Systems- CVS, SVN
Decentralized System- GIT

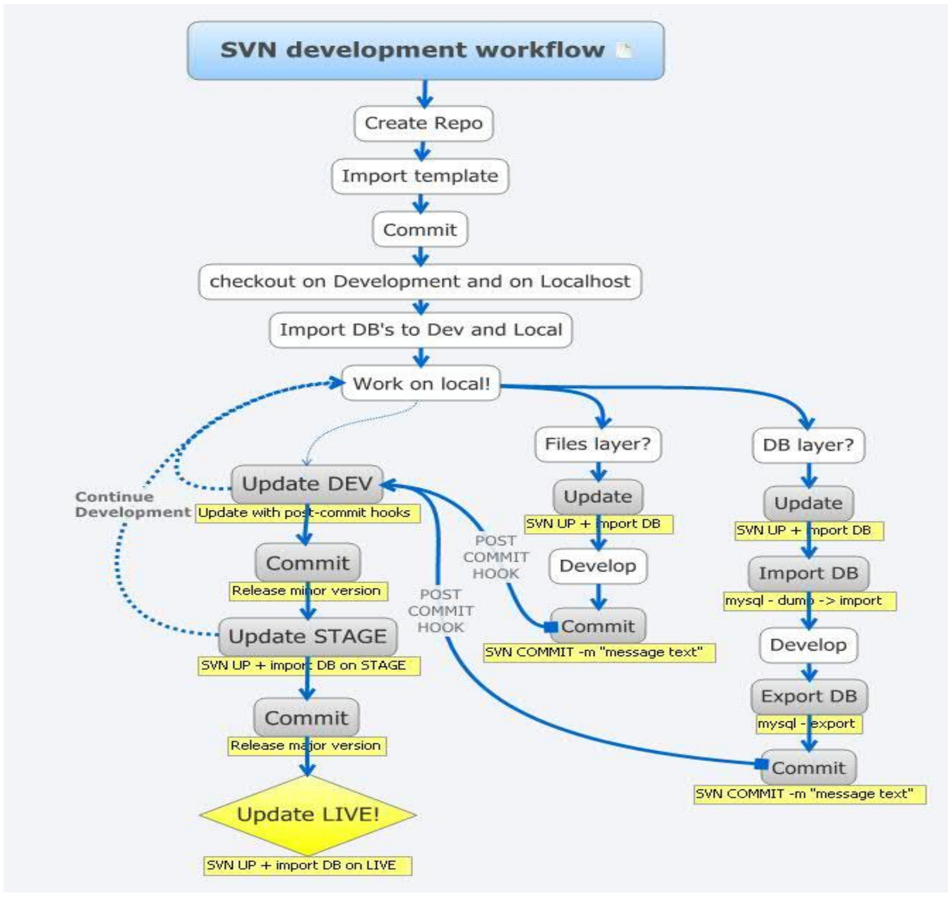
Various Tools used are:

1) CVS
   - CVS was created in the UNIX operating system environment and is available in both Free Software Foundation and commercial versions. It is a popular tool for programmers working on Linux and other UNIX-based systems.
   - CVS works not by keeping track of multiple copies of source code files, but by maintaining a single copy and a record of all the changes.
   - When a developer specifies a particular version, CVS can reconstruct that version from the recorded changes.
   - CVS is typically used to keep track of each developer's work individually in a separate working directory.
   - When desired, the work of a team of developers can be merged in a common repository. Changes from individual team members can be added to the repository through a "commit" command.
   - CVS uses another program, Revision Control System (RCS), to do the actual revision management - that is, keeping the record of changes that goes with each source code file.



2) SVN

- Subversion is very popular due to the fact that it's very easy to understand and very straight forward to work with.
- SVN has one central repository.
- With SVN if the central repository goes down or the code has broken the build then no other developers can commit their code until the repository is fixed.
- More advanced than CVS, less complex than GIT.
- Integrates seamlessly into the Windows shell (i.e. the explorer). This means you can keep working with familiar tools and do not have to change into a different application each time you need the functions of version control.
- Directory versioning:-CVS only tracks the history of individual files, but Subversion implements a "virtual" versioned filesystem that tracks changes to whole directory trees over time. Files *and* directories are versioned.
- Atomic commits:-A commit either goes into the repository completely, or not at all. This allows developers to construct and commit changes as logical chunks.



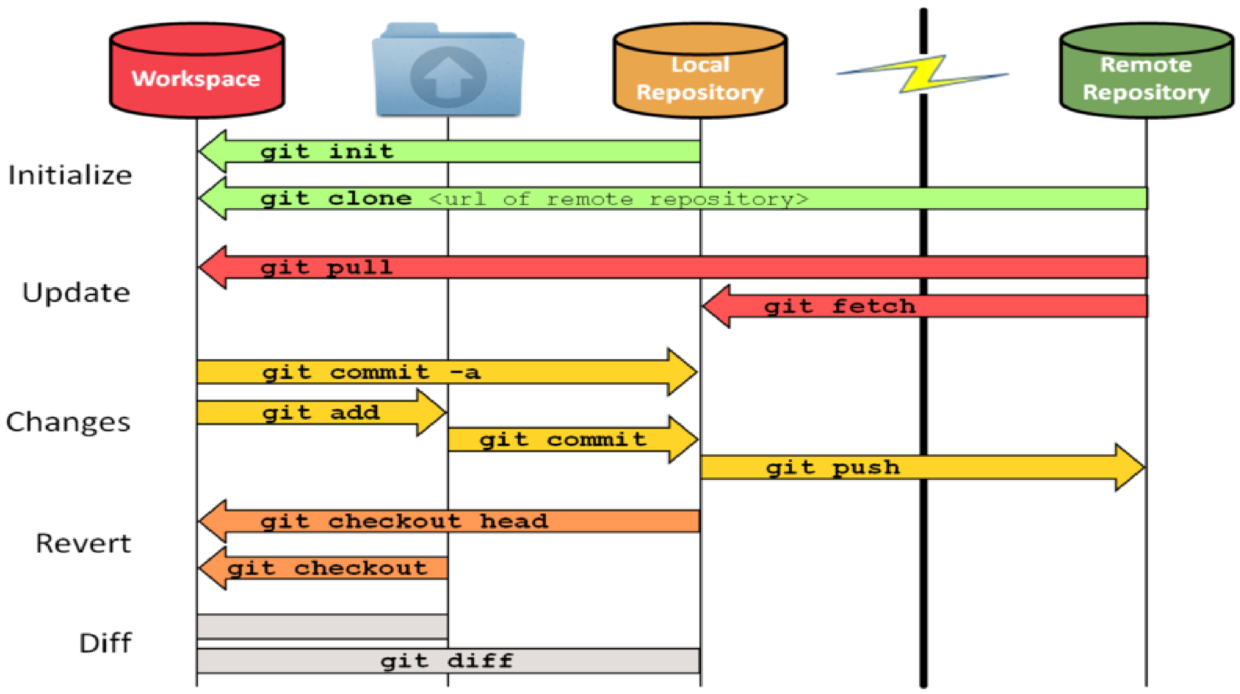
3) GIT

- GIT has multiple repositories, one central repository but each developer has their own repository.
- The benefit of splitting it up into multiple repositories is that there is no longer one single point of failure.
- With GIT as each developer has the own repository it doesn't matter if the master repository is broke we can continue to commit code locally, until the master repository is fixed and then the code can be pushed into the master repository.
- Another benefit of having a local repository is that it is quick, there is no network traffic needed to commit code into source control, developers can work in isolation until ready to push the code to the master repository.
- Creating branches is lightning fast due to Git's branch implementation. In Git, a branch is simply a reference to a commit, where the following commits will be attached.

The workflow is as following:

When working on a project, clone the master repository, this means making a copy of the code at this point in time. This creates a local GIT repository on local machine where one can continue working on a new feature. Create new branches, tags, and continue committing code locally during development.

When a feature is complete, ready to be merged with changes, back into the master repository one needs to push the changes from local repository to the master repository.



Differences between GIT and CVS:

1) **Setting up repository**:

   GIT: Git stores repository in .git directory in top directory of your project.
   CVS: Requires setting up CVSROOT, a central place for storing version control info for different modules. The consequence of that design for user is that importing existing sources into version control is as simple as "git init && git add . && git commit" in Git, while it is more complicated in CVS.

2) **Atomic operations**:

   CVS: Commits and other operations are not atomic in CVS; if an operation on the repository is interrupted in the middle, the repository can be left in an inconsistent state.
   GIT: In Git all operations are atomic: either they succeed as whole, or they fail without any changes.

3) **Naming revisions / version numbers**.

   Version numbers reflects how many time given file has been changed.

   CVS: Changes are per files.
   GIT: Each version of a project as a whole has its unique name given by SHA-1 id. To have version number or symbolic name referring to state of project as a whole you use **tags**. Tags in Git are much easier to use.

4) **Branching**:

   CVS: Branches are overly complicated, and hard to deal with. CVS doesn't have *merge tracking*, so have to be either remembered, or manually tag merges and branching points, and manually supply correct info for "cvs update -j" to merge branches, and it makes for branching to be unnecessary hard to use.
   GIT: Creating and merging branches is very easy; Git remembers all required info by itself. Distributed development naturally leads to multiple branches.

5) **Rename and copy tracking**:

   CVS: File renames are not supported in CVS, and manual renaming might break history in two, or lead to invalid history where you cannot correctly recover the state of a project before rename.
   GIT: Uses heuristic rename detection, based on similarity of contents and filename. You can also request detecting of copying of files. This means that:
   1. When examining specified commit, we would get information that some file was renamed.
   2. Merging correctly takes rename into account
   3. “git blame", the (better) equivalent of "cvs annotate", a tool to show line-wise history of a file contents.

6) **Central Repository:**

   CVS: Maintains a central repository.
   GIT: There is no need to have single central place where you commit your changes. Each developer can have his/her own repository (or better repositories: private one in which he/she does development, and public bare one where she/he publishes that part which is ready), and they can pull/fetch from each other repositories, in symmetric fashion.

7) **Keyword expansion**:
   GIT offers a very, very limited set of keywords as compared to CVS (by default). This is because of two facts: changes in Git are per repository and not per file, and Git avoids modifying files that did not change when switching to other branch or rewinding to other point in history.
8) **Amending commits**:
   In GIT act of publishing is separate from creating a commit, one can change (edit, rewrite) unpublished part of history without inconveniencing other users. The same is not possible in CVS.

Differences between GIT and SVN:

1) **SVN** Repository may be in a location not reachable at the moment, then she/he cannot commit. If a copy of code has to be made, then you have to literally copy/paste it.
   GIT: Local copy is a repository, and a person can commit to it and get all benefits of source control. When connectivity is regained to the main repository, a person can commit against it then.

2) **GIT** has a 'clean' command.
   SVN needs this command, considering how frequently it will dump extra files on your disk.

3) **SVN** creates .svn directories in every single folder.
   GIT only creates *one* .git directory). Every script you write, and every grep you do, will need to be written to ignore these .svn directories. You also need an entire command ("svn export") just to get a sane copy of your files.

4) **SVN** , the user has to keep track when he/she moves or deletes files/folders.
   GIT tracks moved/deleted files.

5) **SVN**, each file & folder can come from a different revision or branch. What this actually means is that there is a million different ways for your local checkout to be completely shuffled up. If some of the files are coming from one place, and some of them from another, the "svn status" might say that everything is normal.
   GIT: Files/folders from different branches are kept separately. If "git status" tells that things are normal, then things really are normal.

Why I chose GIT :-

1) **Speed and Memory**: Saves space and lightning fast.
2) **Work Offline**: Centralized VCS like Subversion or CVS leaves us stranded if not connected to the central repository. With GIT, almost everything is possible simply on local machine: make a commit browse project's complete history, merge or create branches.
3) **Branching**: If a particular feature is disruptive enough that you don't want the entire development team to be affected in its early stages, you can create a branch on which to do this work.
4) **Undo Mistakes**: Reverting a commit, restoring commits is possible on GIT.
5) **Search History:** Find when a function has changed. Detect moved/copied/deleted files.