Version Control Tools and Comparison

Tushaar Gangarapu (156307 15IT117)
January 24, 2018

1. What and Why of Version Control Systems (VCS)

VCS 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. The code for a project, app or software component 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.

2. Various VCS and Comparison

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). The following section elucidates on three VCS- CVS, SVN and Git.

2.1. CVS

Concurrent Versions System (CVS) is a program that lets a code developer save and retrieve different development versions of source code. It also lets a team of developers share control of different versions of files in a common repository of files. 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.

2.1.1 Features of CVS

1. **Client-server model**

CVS uses a client–server architecture: a server stores the current version(s) of a project and its history, and clients connect to the server in order to *check out* a complete copy of the project, work on this copy and then later *check in* their changes.

2. **Open source**

   The source code for development is openly available, can be manipulated, used and inferred for further understanding.

3. **Uses RCS**

   RCS stores the latest version and backward deltas for fastest access and an improved user interface, at the cost of slow branch tip access and missing support for included/excluded deltas.

4. **Specific Features**
   - It can run scripts which you can supply to log CVS operations or enforce site-specific policies.
   - **Client/server CVS** enables developers scattered by geography or slow modems to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. Therefore, the network between the client and the server must be up to perform CVS operations (such as check-ins or updates) but need not be up to edit or manipulate the current versions of the files. Clients can perform all the same operations which are available locally.
   - In cases where several developers or teams want to each maintain their own version of the files, because of geography and/or policy, CVS's **vendor branches** can import a version from another team (even if they don't use CVS), and then CVS can merge the changes from the vendor branch with the latest files if that is what is desired.
   - **Unreserved checkouts**, allowing more than one developer to work on the same files at the same time.
   - CVS provides a flexible **modules database** that provides a symbolic mapping of names to components of a larger software distribution. It applies names to collections of directories and files. A single command can manipulate the entire collection.
   - CVS servers run on most unix variants, and clients for Windows NT/95, OS/2 and VMS are also available. CVS will also operate in what is sometimes called server mode against local repositories on Windows 95/NT.

2.2. SVN

Subversion is a popular open-source version control tool. It is open-source and available for free over the internet. It comes by default with most of the GNU/Linux distributions, so it might be already installed on your system. To check whether it is installed or not use following command.

2.2.1 Features of SVN

1. **Most CVS features**

   CVS is a relatively basic version control system. For the most part, Subversion has matched or exceeded CVS's feature set where those features continue to apply in Subversion particular design.

2. **Directories are versioned**

   Subversion versions directories as first-class objects, just like files.

3. **Copying, deleting, and renaming are versioned**

   Copying and deleting are versioned operations. Renaming is also a versioned operation, albeit with some quirks.

4. **Free-form versioned metadata ("properties")**

   Subversion allows arbitrary metadata ("properties") to be attached to any file or directory. These properties are key/value pairs, and are versioned just like the objects they are attached to. Subversion also provides a way to attach arbitrary key/value properties to a revision (that is, to a committed changeset). These properties are not versioned, since they attach metadata to the version-space itself, but they can be changed at any time.

5. **Atomic commits**

   No part of a commit takes effect until the entire commit has succeeded. Revision numbers are per-commit, not per-file, and commit log message is attached to its revision, not stored redundantly in all the files affected by that commit.

2.2. Git

Git benefits each aspect of your organization, from your development team to your marketing team, and everything in between. By the end of this article, it should be clear that Git isn’t just for agile software development— it’s for agile business. Switching from a centralized version control system to Git changes the way your development team creates software. And, if you’re a company that relies on its software for mission-critical applications, altering your development workflow impacts your entire business.

2.2.1 Features of Git

1. **Feature Branch Workflow**

   One of the biggest advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge. This facilitates the feature branch workflow popular with many Git users.

2. **Distributed Development**

   In SVN, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits.

3. **Pull Requests**

   Many source code management tools such as Bitbucket enhance core Git functionality with pull requests. A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of changes, but also lets developers initiate discussions around their work before integrating it with the rest of the codebase.

4. **Community**

   In many circles, Git has come to be the expected version control system for new projects. If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.

5. **Faster Release Cycle**

   The ultimate result of feature branches, distributed development, pull requests, and a stable community is a faster release cycle. These capabilities facilitate an agile workflow where developers are encouraged to share smaller changes more frequently. In turn, changes can get pushed down the deployment pipeline faster than the monolithic releases common with centralized version control systems.

Feature branches lend themselves to rapid prototyping. Whether your UX/UI designers want to implement an entirely new user flow or simply replace some icons, checking out a new branch gives them a sandboxed environment to play with. This lets designers see how their changes will look in a real working copy of the product without the threat of breaking existing functionality.