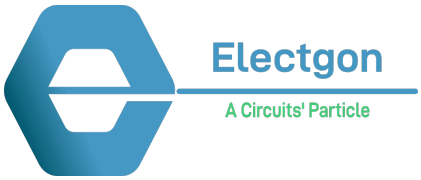

Version Management Tools

ELECTGON
www.electgon.com
contact@electgon.com

16.12.2017

Contents

1 Introduction

1.1 What is Repository

In software development, a repository is a central file storage location. It is used by version control systems to store multiple versions of files. While a repository can be configured on a local machine for a single user, it is often stored on a server, which can be accessed by multiple users.[1]

1.2 Version Control systems

Version control is used to manage multiple versions of computer files and programs. A version control system, or VCS, provides two primary data management capabilities. It allows users to

1) lock files so they can only be edited by one person at a time, and
2) track changes to files.[2]

1.3 Common Terminologies

To understand version control systems, it may be useful to clarify common terminologies that are frequently used within a version control system.

Branch: For multiple design files or different configuration of your design, it is useful to include design files in categories. For instance you may have some design files implemented in HTML format that are to be used for building web interface of your system. You may have also design files described in python script that handle interaction between web objects and database. So maintaining each design files in specific branch keeps organization of your system design files specially if there are different developers working on each branch.

Baseline: When development phase reaches accepted conditions and your system is ready for release, last design files are labeled with a release number to indicate that these numbers represent stable and working version of the system. This release then becomes a baseline of your development system. So it reflects mainly different release of the system.

Checkout: developer working an some design files, has to submit his work to the main system. This submission is called checkout.

Clone: it is the action of copying the reference repository into a local machine so that developer can add his work to that copied repository. This cloning can take place also within the same machine i.e. this cloning step is not limited to a server-local machine connection, you can clone original repository from one directory to another.

Commit: adding the developed design files to the repository is called commit. It means also that changes made to existing files are added also at this step. Making a commit to the

repository means updating included files in the repository so new commit means new update took place on the repository.

Merge: for a design file that has been changed by two different developers, merging means that both developers have to include their changes together in the final design file. Merging may also be used to merge two or more branches together so that the final resulted branch will include all design files existing in each branch,

Push: it means adding the content or changes made in a repository to another remote repository.

Pull: it means getting the content or changes made in a remote repository to the active repository.

Head: for each commit process, there is a pointer points to last updated status and files of the repository. This pointer is called Head and in some version control system it can switch freely between new and old commits.

Tag: to remark important commits or releases or baseline, it is usually marked by a tag so that any used can switch easily between different commits by knowing name of the tag assigned to the target commit.

Trunk: in some version control system, trunk refers to the main or original branch, any other branch in the repository are derived from this main branch.

Figure 1 shows some of these basic terminologies that shape out how a version control repository may look like.

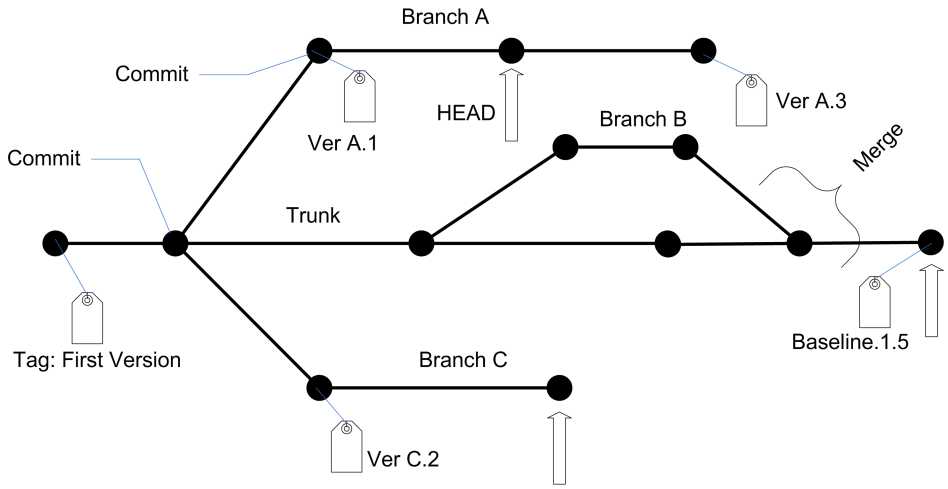

Figure 1: Structure of a Repository

2 Topologies of Repository Systems

There are currently many version control systems in the market. Some of them are open source, others are proprietary. These tools, however is classified into two types:

2.1 Centralized Version Control systems (CVCS)

First let's look at CVCS as shown in figure 2. The central repository serves as the hub and developers act as separate spokes. All work goes through the central repository. This makes version control easy and sharing difficult.

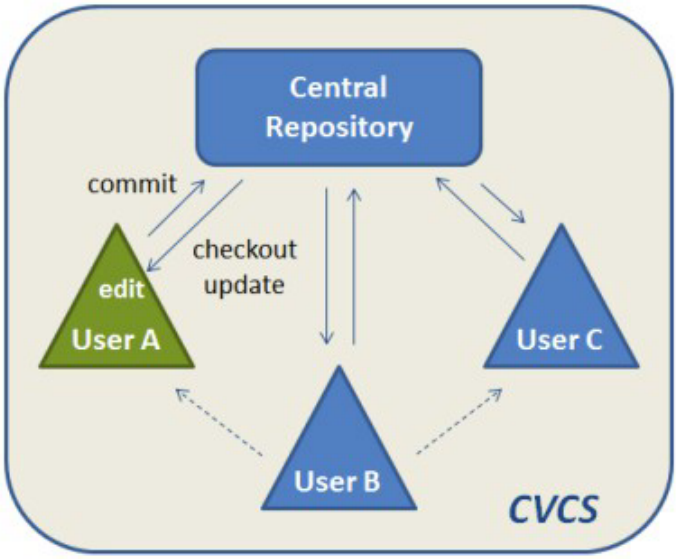

Figure 2: CVCS System [3]

When you're working with a centralized version control system, your workflow for adding a new feature or fixing a bug in your project will usually look something like this:

- Pull down any changes other people have made from the central server.
- Make your development, and make sure they work properly.
- Commit your changes to the central server, so other programmers can see them.

2.2 Distributed Version Control Systems (DVCS)

With DVCS there is more interaction directly between developers as shown in figure 3. Systems are designed with the intent that one repository is as good as any other, and that merges from one repository to another are just another form of communication.

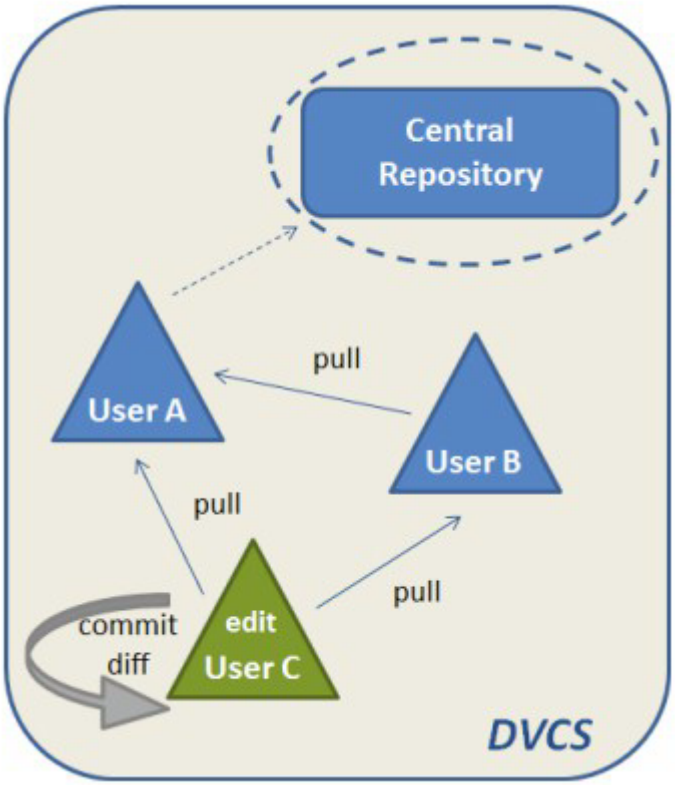

Figure 3: DVCS System [3]

When you work with distributed version control system, your workflow will be like this:

- Cloning the original repository, you will have local repository then.
- Make your development, and make sure that they are working properly.
- Commit your changes to your local repository.
- If you want to update other repository, you push your updated repository to the target repository.

2.3 Comparison Between CVCS and DVCS

One important step to be able to find out which tool can facilitate your work, is to understand or be aware of topology of this tool. This can give overview about mechanism of storing and sharing source code files. Then you can decide if this mechanism is suitable or not. Tables 1 shows quick comparison between the two topologies

CVCS	DVCS
A repository contains all the information and history about a project	A project repository may be cloned, resulting in multiple equivalent copies.
Development cycle, per developer, is typically: 1. Check out (copy) working files from the repository. 2. Make changes to the working files. 3. Update and possibly merge from the repository to get any changes made by others. 4. Commit the new version back to the repository.	First, Repositories may change independently and be synchronized using push or pull operations. Second, depending on the change, you can update or merge (Updates happen when there's no ambiguity. Merges are needed when we have conflicting changes.)
Previous versions of the project may be easily recovered.	Previous versions of the project may be easily recovered.
Multiple branches of the project may be stored simultaneously, enabling parallel development	Branches is also available for each user.
Merging may result in conflicts that must be resolved manually. For example, when two developers make incompatible changes to the same section of code, the first to commit will succeed with no errors, while the second will get a conflict because the code he checked out is not the current code in the repository.	Update or merge is done after push or pull the repository, if there is conflict it has to be resolved manually preventing the developer from overwriting others work.
Each developer has his own copy of working files for either the entire source tree or just sub-parts	Each developer has his own local copy of the entire repository including history and working files.
Requires less disk space.	Requires more disk space.
There is a single master repository by definition.	Developers must agree on a "master" repository where all changes get integrated.
Developers must contact master repository to get information about target branch.	Developers can cheaply switch between branches of development using local repository.
Central machine has all versions	Backup for local machine is advised
Central machine has the latest "stable" release	No latest version, you may get confused about who is having the latest version

Table 1: Comparison between CVCS and DVCS Systems

3 Tools Used in Each Topology

Here are most common softwares used as version control systems

CVCS: Open Source: CVS – Subversion (SVN) - Vesta
Proprietary: IBM Rational ClearCase – Perforce – Team Foundation Server.

DVCS: Open Source: Bazaar – Git – Mercurial.

Proprietary: BitKeeper – Sun Workshop.

4 Tools Comparison

We will try to come up with best CVCS/DVCS powerful tools. This interrogation is done against Open Source tools.

In table 2 a comparison between some important features of CVCS open source tools:

	Maintainer	Development Status	Platforms supported	Web interfaces	Standalone GUI	Integration with other tools
CVS	CVS team	maintained but new features not added	Unix, Windows	CVS web, ViewVC	TkCVS WinCVS	Eclipse Emacs
Subversion	Apache Software Foundation	actively developed	Unix, Windows	WebSVN ViewSVN Trac	TortoiseSVN RapidSVN SourceTree	Eclipse Emacs Visual Studio
Vesta	Kenneth Schalk Tim Mann	latest release February 15, 2009	Unix	VestaWeb	No	No

Table 2: Comparison between open source CVCS tools [4]

Based on this comparison we can find that SVN is the one most used. Actually CVS suffered a number of limitations, such as being unable to move or rename files. SVN was initially developed as an alternative to CVS. Regarding Vesta it suffers also from ability of merging files between different branches, in addition to it hasn't updated since 2009 which may indicates no more enhancement applied for this software.

Now, we can examine the same comparison for DVCS tools.

	Maintainer	Development Status	Platforms supported	Web interfaces	Standalone GUI	Integration with other tools
Bazzar	Canonical Ltd	limited development	Unix, Windows	Trac	Olive, Bazaar Explorer	Eclipse, Visual Studio
Git	Junio Hamano	actively developed	Unix, Windows	GitHub, Trac and many other web interfaces	Git-gui, gitbox, source tree	Eclipse, Visual Studio, Emacs, Komodo IDE
Mercurial	Matt Mackall	actively developed	Unix, Windows	Trac	Hgk, sourceTree	Eclipse, Visual Studio, Emacs, Komodo IDE

Table 3: Comparison between open source DVCS tools [4]

We can find then that these tools are similar in these features. However still there is some differences can be summarized as follows:

1. Performance : In terms of raw performance, Git leads almost all benchmarks followed by Mercurial and then by Bazaar. A detailed benchmark can be shown in table 4

	Git 1.5.4.3	Bazaar 1.3.1	Mercurial 0.9.5
Initialization	0.086s	0.334s	0.137s
Adding	14.269s	4.852s	2.526s
Commit	10.263s	43.968s	30.890s
Commit (small)	0.397s	9.010s	1.913s
Diff (large)	24.425s	51.158s	37.846s
Diff (no changes)	0.343s	47.448s	1.340s
Status (no changes)	1.230s	4.027s	1.077s

Table 4: Advanced Comparison between DVCS tools[5]

2. Final Repository Size: Again Git wins here, with the least size of the Final Repository. It is followed by Bazaar and then by Mercurial.

Git	Bazaar	Mercurial
92 MB	112 MB	179 MB

Table 5: Comparison Between Repositories Size

3. Design Differences: Mercurial and Bazaar are FILE LEVEL VCS as they store versions of files but GIT is a CONTENT LEVEL VCS i.e. it stores delta of content, not the file itself.

4. Repository Hosting: For open-source and private repository hosting both Mercurial and Git are way ahead of Bazaar in-terms of number of Source code hosting providers available. When there are more than 9 hosting providers for Mercurial and about 8 hosting providers for Git, only 3 hosting providers are available for Bazaar.[6]

So we can recommend here that Git is best choice if we like DVCS approach.

5 Nominated Tools in Scope

So we can nominate from previous comparison SVN as CVCS tool and Git as DVCS tool, now if we need to come up with most appropriate tool, several aspects we have to look for.

– What is better to work with CVCS or DVCS.
– More about nominated tools from each topology.
– What experts say about these tools
– In Which applications our repository will be used.

So this decision is taken by the administrator of the system. Previous comparison was a trial to shed the light on possible difference between these tools. What still can be shown here is a light detail about working with the nominated tools SVN and Git.

5.1 SVN

With Subversion, you have access to a repository inside which you can insert your project files. Subversion keeps track of all the changes you make to your files; all the different versions of the files you submit correspond to revisions. You usually access the most recent revision, use it, make change to it and then submit a new version. You can also obtain files in the state they were at any past revisions.

Directories in your Repository

One way to use this directory structure is as follow. The software product is developed under the trunk. When the software is ready for an official release, you copy it to a branch (e.g. version 1.0) and to a tag (version 1.00). This is the tagged version that your clients will use. While the development team continue to develop the new features on the main project (under the trunk), the quality assurance team maintain the version in the branch in order to fix the small bugs. When enough bugs have been corrected, you are ready to generate a new tag (version 1.01,...). In parallel, new release of the software will generate new branches (version 2.0, 3.0,...). [7]

Working with SVN

STEP 0. Creating a new directory: This can be done simply by

```
> svn import
```

STEP 1. Checking out a directory: This can be done by

```
> svn checkout <directory of the repository> <copy name>
```

STEP 2. Editing: you can add, delete, copy or move files.
STEP 3. Committing your changes: Using

```
> svn commit
```

The modified files will be uploaded to the server and a new revision will be created that other users can, in turn, check out.[7]

5.2 Git

You can get a Git project using two main approaches. The first takes an existing project or directory and imports it into Git. The second clones an existing Git repository from another server.

Initializing a Repository in an Existing Directory

You need to go to the project's directory and type

```
> git init
```

Cloning an Existing Repository

This can be done by typing

```
> git clone <directory of the repository>
```

Tracking New Files

In order to begin tracking a new file, you use the command

```
> git add <file name>
```

Staging Modified Files

In Git, Files that are modified are needed to be Staged first before committing this can be done by

```
> git status
```

Committing Your Changes

The simplest way to commit is to type

```
> git commit
```

Edit Files

In Git you can also remove files

```
> git rm
```

or Move files

```
> git mv
```

Integrate your work with other repository

This can be done using Push or Pull

to pull a repository from other machine

```
> git fetch [remote–name]
```

to push a repository into other machine

```
> git push <origin master>
```

[8]

6 Conclusion

At this point we can find out that both tools (SVN and Git) have easy commands to work with. The main difference is structure or topology that each tool is following (CVCS or DVCS). As mentioned in previous section, decision of which tool to choose depends on the user system and business requirements. These requirement may specify a need to store and track development files, with options to commit process management files (Minutes of Meeting, Plans, Specifications, ...) Other business plans that may arise, like Bug tracking environment, automation environment, etc shall be taken into consideration also. These environments will be in interact with development files which are kept in version management tool.

By keeping these business requirements in mind, you can examine the following feature in the nominated tool to make sure that it can fulfill your requirements:

– Stable: no bugs or no severe complaints observed on this tool.

– High Performance: in terms of minimizing time and effort needed for committing or merging different version files.

– Efficiency of Use: to be customizable, simple and easy to work with.

– Ability to be later on integrated with bug tracking systems/Automation tools.

Bibliography

[1] Source: ”http://www.techterms.com/definition/repository”.

[2] Source: ”http://www.techterms.com/definition/version_control”.

[3] Source: ”http://blog.appfusions.com/cvcs-vs-dvcs-and-the-pros-and-cons-of-dvcs-git”.

[4] Source: ”http://en.wikipedia.org/wiki/Comparison_of_revision_control_software”.

[5] Source: ”https://git.wiki.kernel.org/index.php/GitBenchmarks#bzr.2C_git.2C_and_hg_performance_on_the_Linux_tree”.

[6] Source: ”http://www.techtatva.com/2010/09/git-mercurial-and-bazaar-a-comparison”.

[7] Source: ”http://laganiere.name/subversionTut/index.shtml”.

[8] Source: ”http://git-scm.com/doc”.