Version Control System (VCS)

A VCS is a tool for managing a collection of program code that provides you with three important capabilities: reversibility, concurrency, and annotation. It is synonymously also known as Source Code Management (SCM). VCS is a combination of technologies and practices for tracking and controlling changes to a project's files, in particular to source code, documentation, and web pages.

VCS is universal as it helps with almost all aspect of a project: inter-developer communications, release management, bug management, etc. The version control system acts as a central coordinating force among all of these areas. The core of version control is change management. Change Management deals with identifying each discrete change made to the project's files, annotating those changes with metadata (e.g. date and author) and then reproduce these facts whenever desired. It is a communications mechanism where a change is the basic unit of information.

How do we normally work without VCS?

- Keep backups in folders (V1, V2- released, V3, V3-stable,…) or tarballs (v1.tar.gz,v2.tar.gz,..)

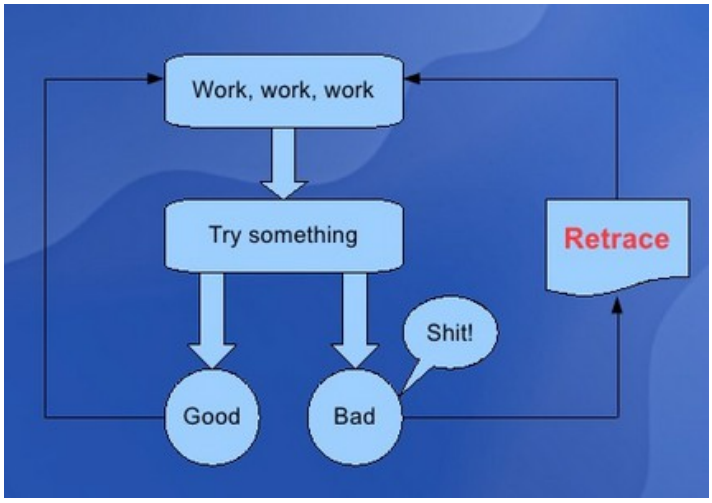

Fig: Work Cycle without VCS

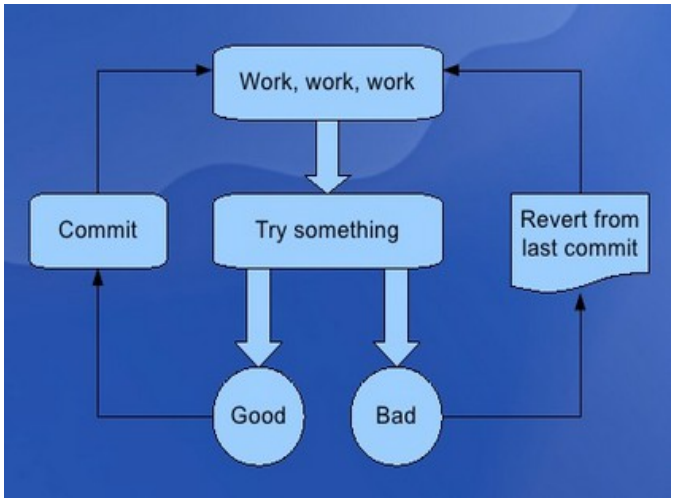

Fig: Work Cycle with VCS

We should follow the principal of versioning everything. Apply versioning not only the project's source code under version control, but also its web pages, documentation, FAQ, design notes, and anything else might undergo an update/edit. Keep them right next to the source code, in the same repository tree. Any piece of information worth writing down is worth versioning—that is, any piece of information that could change. Things that don't change should be archived, not versioned. For example, an email, once posted, does not change; therefore, versioning it wouldn't make sense (unless it becomes part of some larger, evolving document).

BASIC TERMS

repository

The set of files or directories that are under version control are more commonly called a repository. It is a database in which changes are stored. Some VCS can centralized: there is a single, master repository, which stores all changes to the project. Others are decentralized/distributed: each developer has his own repository, and changes can be swapped back and forth between repositories arbitrarily.

Revision, change, changeset

A "revision" is usually one specific incarnation of a particular file or directory. For example, if the project starts out with revision 2 of file File1, and then someone commits a change to File1, this produces revision 3 of File1. Some systems also use "revision", "change", or "changeset" to refer to a set of changes committed together as one conceptual unit. It tells precisely about exact points in time in the history of a file or a set of files (i.e., immediately before and after a bug is fixed). When one talks about a file or collection of files without specifying a particular revision, it is generally assumed to be the latest revision available.

commit

To make a change, i.e. to store a change in the version control database in such a way that it can be incorporated into future releases of the project. "Commit" can be used as a verb or a noun.

Log message

A bit of commentary attached to each commit, describing the nature and purpose of the commit. Log messages are among the most important documents in any project: they are the bridge between the highly technical language of individual code changes and the more user-oriented language of features, bugfixes, and project progress

update

To ask that others' changes (commits) be incorporated into your local copy of the project; that is, to bring your copy "up-to-date". This is a very common operation; most developers update their code several times a day, so that they know they're running roughly the same thing the other developers are running, and so that if they see a bug, they can be pretty sure it hasn't been fixed already.

checkout

The process of obtaining a copy of the project from a repository. A checkout usually produces a directory tree called a "working copy", from which changes may be committed back to the original repository. In some decentralized version control systems, each working copy is itself a repository, and changes can be pushed out to (or pulled into) any repository that's willing to accept them.

working copy

A developer's private directory tree containing the project's source code files, and other documents. A working copy also contains a little bit of metadata managed by the version control system, telling the working copy what repository it comes from, what "revisions" of the files are present, etc.

diff

A textual representation of a change. A diff shows which lines were changed and how.

tag

A label for a particular collection of files at specified revisions. Tags are usually used to preserve interesting snapshots of the project. For example, a tag is usually made for each public release, so that one can obtain, directly from the version control system, the exact set of files/revisions comprising that release. Common tag names are things like Release_1_0, Delivery_00456, etc.

branch

A copy of the project, under version control but isolated, so that changes made to the branch don't affect the rest of the project, and vice versa, except when changes are deliberately "merged" from one side to the other. Branches are also known as "lines of development". Branches offer a way to isolate different lines of development from each other. For example, a branch can be used for experimental development that would be too destabilizing for the main trunk. Or conversely, a branch can be used as a place to stabilize a new release. During the release process, regular development would continue uninterrupted in the main branch of the repository; meanwhile, on the release branch, no changes are allowed except those approved by the release managers. This way, making a release needn't interfere with ongoing development work.

Merge (a.k.a. port)

To move a change from one branch to another. This includes merging from the main trunk to some other branch, or vice versa. In fact, those are the most common kinds of merges; it is rare to port a change between two non-main branches. "Merge" has a second, related meaning: it is what the version control system does when it sees that two people have changed the same file but in non-overlapping ways. Since the two changes do not interfere with each other, when one of the people updates their copy of the file (already containing their own changes), the other person's changes will be automatically merged in. This is very common, especially on projects where multiple people are hacking on the same code. When two different changes do overlap, the result is a "conflict".

Conflict

What happens when two people try to make different changes to the same place in the code. All version control systems automatically detect conflicts, and notify at least one of the humans involved that their changes conflict with someone else's. It is then up to that human to resolve the conflict, and to communicate that resolution to the version control system.

lock

A way to declare an exclusive intent to change a particular file or directory. Not all version control systems even offer the ability to lock, and of those that do, not all require the locking feature to be used. This is because parallel, simultaneous development is the norm, and locking people out of files is (usually) contrary to this ideal.

Version control systems that require locking to make commits are said to use the lock-modify-unlock model. Those that do not are said to use the copy-modify-merge model.

Types of VCS: Centralized vs Distributed

Currently, the most popular version control system in use is Subversion, which is considered a centralized version control system. The main concept of a centralized system is that it works in a client and server relationship. The repository is located in one place and provides access to many clients. It’s very similar to FTP in where you have an FTP client which connects to an FTP server. All changes, users, commits and information must be sent and received from this central repository.

The primary benefits of Subversion are:

- It is easy to understand and get started.
- You have more control over users and access (since it is served from one place).
- More GUI & IDE clients (Subversion has been around longer).

The main drawbacks of Subversion are:

- Dependent on access to the server.
- Hard to manage a server and backups.
- It can be slower because every command connects to the server.
- Branching and merging tools are difficult to use.

Decentralized/Distributed systems are a newer alternative for traditional centralized VCS. In distributed version control, each user has their own copy of the entire repository, not just the files but the history as well. Think of it as a network of individual repositories. The most popular VCS in this category is Git.

The primary benefits are:

- It is fast. More powerful and detailed change tracking, which means less conflicts.
- No server necessary – all actions except sharing repositories are local (commit offline).
- Branching and merging is more reliable, and therefore used more often.

Some drawbacks:

- The distributed model is harder to understand and not much GUI clients (as it is new).
- The revisions are not incremental numbers, which make them harder to reference.
- It can be easier to make mistakes until you are familiar with the model.

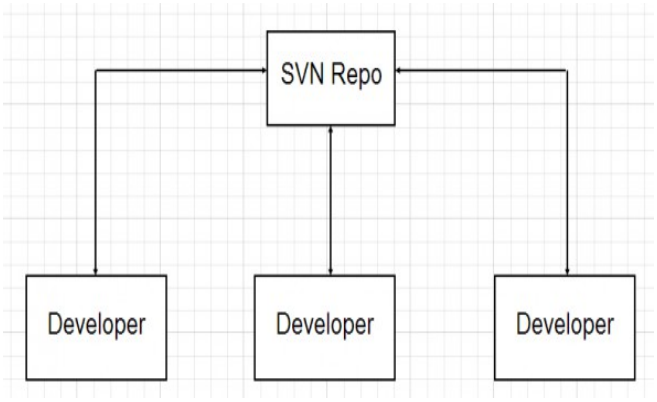

Fig: Centralized VCS

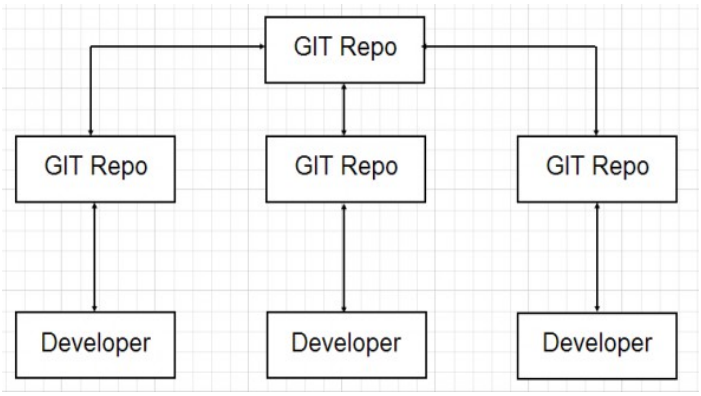

Fig: Decentralized VCS

References:

http://www.slideshare.net/xSawyer/source-code-management-systems

http://zeroturnaround.com/rebellabs/devprod-report-revisited-version-control-systems-in-2013/#!/

http://producingoss.com/en/vc.html

http://guides.beanstalkapp.com/version-control/intro-to-version-control.html

http://agile.dzone.com/articles/version-control-git-vs-svn

http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html

http://wiki.spheredev.org/Git_for_the_lazy

http://git-scm.com/book/en/Getting-Started-About-Version-Control