

# Distributed Version Control - better than you think it is

Gary Pampará <gpampara@gmail.com>

June 2, 2009

## Outline

Introduction  
Centralized systems  
Distributed systems  
Git  
Resources  
Demo  
Conclusion

Introduction

Centralized systems

Distributed systems

Git

Resources

Demo

Conclusion

- ▶ Version control is stated as being a tool that is a requirement in the software industry.
- ▶ What is it exactly anyways?

- ▶ Version control is stated as being a tool that is a requirement in the software industry.
- ▶ What is it exactly anyways?

### Version control is defined to be:

The management of changes to documents, programs, and other information stored as computer files.

- ▶ Why do we need it?

- ▶ Why do we need it?
- ▶ Should version control be something that helps?

- ▶ Why do we need it?
- ▶ Should version control be something that helps?
- ▶ How can I maximize the the benefit I get from a version control system?

# What options do we have?

- ▶ What Source Code Management (SCM) options are available?

# What options do we have?

- ▶ What Source Code Management (SCM) options are available?
- ▶ There are two types of version control:
  - ▶ Centralized version control
  - ▶ Decentralized version control

- ▶ I have a system that manages differences of files. How does that actually help me?

- ▶ I have a system that manages differences of files. How does that actually help me?
- ▶ Tarballs & patches would work as well - do I really need a system to do it?

- ▶ I have a system that manages differences of files. How does that actually help me?
- ▶ Tarballs & patches would work as well - do I really need a system to do it?
- ▶ Yes, we **DO** need a system and a **GOOD** one at that.

- ▶ There is a single location to where information is stored.
- ▶ All versioned information is public.
- ▶ The notion of “commit access” for developers.
- ▶ Free hosting services: Sourceforge.net etc.
- ▶ You **have to be connected** to work.
- ▶ Primitive synchronization schemes:
  - ▶ Locking of files

- ▶ There is a single location to where information is stored.
- ▶ All versioned information is public.
- ▶ The notion of “commit access” for developers.
- ▶ Free hosting services: Sourceforge.net etc.
- ▶ You **have to be connected** to work.
- ▶ Primitive synchronization schemes:
  - ▶ Locking of files
  - ▶ Version merging

- ▶ There is a single location to where information is stored.
- ▶ All versioned information is public.
- ▶ The notion of “commit access” for developers.
- ▶ Free hosting services: Sourceforge.net etc.
- ▶ You **have to be connected** to work.
- ▶ Primitive synchronization schemes:
  - ▶ Locking of files
  - ▶ Version merging
- ▶ Generally the maintainer of a project is responsible for merging patches in.

## The Concurrent Versions System (CVS):

- ▶ Developed in the 1980's
- ▶ Applies versioning to files separately
- ▶ Isolates conflicts by allowing people to lock files
- ▶ If files are not locked, funnies can happen with simultaneous commits.
- ▶ Most people have some experience with CVS :(
- ▶ Terrible CVS directories EVERYWHERE!

## Subversion (aka SVN):

- ▶ Developed by CollabNet in 2000
- ▶ Quite popular
- ▶ Project aims to correct and fix bugs in CVS and provide the features missing in CVS. Basically: "SVN is CVS done right."
- ▶ SVN caters for atomic commits
- ▶ Rename / copy / remove / move preserved in history
- ▶ Operates on files individually
- ▶ Handles binary data better
- ▶ .svn directories EVERYWHERE!

Other centralized systems exist, all with variations to the functionality in CVS / SVN but they all have the same model:

- ▶ Perforce
- ▶ CVSNT (It's horrible!)
- ▶ SVK (Adds some distribution concepts to SVN as an additional layer - hungry for disk space)
- ▶ Other systems that are not worth mentioning
- ▶ ...
- ▶ Oh, and Visual Source Safe. This software is like deliberately and systematically destroying your will to live by allowing yourself to be influenced by morons. Writing code on paper and using a series of post-it notes to manage your diffs is a better system.

- ▶ What happens to a project when there are 10000+ people wanting to commit?
- ▶ Not always easy to explain
- ▶ No central location to stop data. No single location is “better” than another location.
- ▶ Ability to work offline without needing to have commit rights (commit people, commit!). There are other more important things to discuss :)
- ▶ Branching is strongly encouraged (unlike centralized SCMs)

- ▶ Everyone should have their own branch to work on.
- ▶ No need to worry about locking files because you are working locally.
- ▶ Can develop and test features independently. Main development can still continue on the HEAD.
- ▶ People can decide to obtain changes from trusted sources. You can't simply trust all people :P

A smaller suite of distributed version control systems exist:

- ▶ Bazaar (Written in Python, Pyrex, C)
- ▶ Codeville (Written in Python - Bittorrent uses this)
- ▶ Darcs (Written in Haskell)
- ▶ Git (Written in C)
- ▶ GNU Arch (Written in C, Shell scripts)
- ▶ Mercurial (Written in Python)
- ▶ Monotone (Written in C++)

## The reason we are here

- ▶ Credit must be given to CVS:
  - ▶ Credit is negative
  - ▶ WWCVSND - What would CVS never do
- ▶ SVN is the most pointless project ever created
- ▶ SVN's slogan is: "CVS done right"

## The reason we are here

- ▶ Credit must be given to CVS:
  - ▶ Credit is negative
  - ▶ WWCVSND - What would CVS never do
- ▶ SVN is the most pointless project ever created
- ▶ SVN's slogan is: "CVS done right"
- ▶ How is that possible considering the damage CVS has done?

## The reason we are here

- ▶ Credit must be given to CVS:
  - ▶ Credit is negative
  - ▶ WWCVSND - What would CVS never do
- ▶ SVN is the most pointless project ever created
- ▶ SVN's slogan is: "CVS done right"
- ▶ How is that possible considering the damage CVS has done?
- ▶ Many centralized systems brag at how cheap it is to create a branch. How easy is it to merge a branch?
  - ▶ That's meaningless!
  - ▶ Real question: How simple is it to merge a branch?

## The reason we are here

- ▶ Credit must be given to CVS:
  - ▶ Credit is negative
  - ▶ WWCVSND - What would CVS never do
- ▶ SVN is the most pointless project ever created
- ▶ SVN's slogan is: "CVS done right"
- ▶ How is that possible considering the damage CVS has done?
- ▶ Many centralized systems brag at how cheap it is to create a branch. How easy is it to merge a branch?
  - ▶ That's meaningless!
  - ▶ Real question: How simple is it to merge a branch?
  - ▶ Who has successfully merged a CVS or SVN branch? Who actually enjoyed the process?

BitMover changed the licensing for BitKeeper. The Linux source tree needed a new system.

Linus examined many systems, most of which could be discarded based on the requirements below:

BitMover changed the licensing for BitKeeper. The Linux source tree needed a new system.

Linus examined many systems, most of which could be discarded based on the requirements below:

- ▶ If not distributed: not interested

BitMover changed the licensing for BitKeeper. The Linux source tree needed a new system.

Linus examined many systems, most of which could be discarded based on the requirements below:

- ▶ If not distributed: not interested
- ▶ If performs badly: not interested

BitMover changed the licensing for BitKeeper. The Linux source tree needed a new system.

Linus examined many systems, most of which could be discarded based on the requirements below:

- ▶ If not distributed: not interested
- ▶ If performs badly: not interested
- ▶ If cannot 100% guarantee that what is put into the system is not what is retrieved: not interested. How many times has code “disappeared” in your projects?

Monotone looked very promising but the performance was terrible. Linus believed that he could write a system in two weeks, the result was Git. Many flow ideas were taken from BitKeeper.

Git strives to provide:

- ▶ Reliability
- ▶ Highly performant
- ▶ Distributed
- ▶ Manages content

Also provides:

- ▶ Strong support for non-linear development
- ▶ Cryptographic authentication of history

- ▶ Git tracks content. It does not care about files or directories.
- ▶ Git manages the entire project scope, including the history.
- ▶ Git will notice corruptions on the data, guaranteed. Uses a SHA-1 hash to ensure data validity.
- ▶ Because of the SHA-1 hash, you can obtain a tree from someone you don't trust and have the faith that the tree you get is exactly how you expect it to be.

I maintain Cllib using Git - it has been amazing so far.  
Here are a list of some of the projects that are using Git, some are rather popular:

- ▶ Git (shows trust)
- ▶ Linux Kernel
- ▶ Perl (including ALL history for the past 20 years)
- ▶ Gnome
- ▶ Ruby on Rails
- ▶ Google Android
- ▶ Wine
- ▶ X.org
- ▶ Prototype
- ▶ Qt (Recently released the toolkit on gitorious.org)
- ▶ C++ Boost library

Some general resources:

- ▶ <http://git-scm.org>
- ▶ <http://git-scm.com/documentation>
- ▶ <http://www.kernel.org/pub/software/scm/git/docs/everyday.html>
- ▶ <http://github.com>
- ▶ <http://gitorious.org>

The only real way to find out how Git works is to play with it :)  
Lets have a look - here you all drive, I'll do the work

Git and other distributed version control systems are much better than what people give them credit for. It does require a brain shift to understand the rationale behind them but it's worth the time investment.

The excuse “just because” is not a valid one! Use it. It'll make your life much simpler and will mean that a tool is an asset, not something you just “have to use”.

Thank you :) Any questions?