**Git : Version Control System ( VCS )**

Managing Source code ( 90-95 %) => Source Code Management ( SCM )

1. Keeps track of changes especially text changes

2. version1, version2, version3 ......

examples of version control ( non-source code )

=> File naming (rat.gif, )

=> Microsoft Word's Track changes

=> Adobe Photoshop's History

=> Undo - Control + z (on windows)

**History Of Git :**

**Source Code Control System (SCCS)** : 1972 closed source, free with Unix ( only single file )

=> Saves the snapshot of what the changes were

**Revision Control System (RCS)** : 1982, open source ( works only with single file )

**Concurrent version system (CVS)**: 1986-90, open source, Put file on a remote server (more than one user can work on a file concurrently)

**Apache Subversion (SVN)** : 2000, open source.

Saving of Non text files, Watching the files in the directory, Track the history of Directories.

**BitKeeper SCM** : 2000, Closed source, proprietary, Distributed version Control

" Community Version was free "

1. used for source code ofLinux Kernel , Controversioal to use Propreity SCM for an open source Project (in April 2005 BitKeeper Community Version Was stopped for free )

**Git is born ( April 2005 ) :**

Git was created by Linus Torvalds ( Created Linux )

Replacement for BitKeeper to manage Linux Kernel Source Code.

Distributed Vesion Control & Open source and Free software

Compatible with unix like systems ( Linux, Mac OS X, Solaris ) & windows

Faster than other SCM's (100x in some cases ).

Better safeguards against data Corruption.

Git became a hit, as people discovered the power of Distributed Version Control as they got used to all of Git's nice features, Git has experienced an explosion in popularity. Now there is no official statistics on this but to give you an example,

GitHub launched in 2008 to host Git source code repositories. In 2009, there were over 50,000 repositories with a 100,000 users.

In 2011 just two years later, there were 2 million repositories with over a million users.

So you can see this rapid growth in just two years.

**Distributed Version Control :**

we talked about SCSS, RCS, CVS, and SVN, four of the most popular version control systems of the past. And all four of these **use a central code repository model**, that is that there is one central place where you store the master copy of your code, and when you're working with the code you check out a copy from the master repository. You work with it make your changes, and then you submit those changes back to the central repository.

Other users can also work from that repository submitting their changes. And it's up to us as users to keep up-to-date with whatever is happening in that central code repository, to make sure that we pull down and update any changes that other people have made.

Git doesn't work that way, **Git is Distributed Version Control**, different users --or teams of users--each maintain their own repositories instead of working from a central repository. And the changes are stored as **change sets** or **patches**, and we're focused on **tracking changes** not the versions of the document

Now that's a subtle difference, you may think, well, CVS and SVN those track changes too, they don't. They track the changes that it takes to Git from version-to-version of each of the different files or the different states of a directory. Git doesn't work that way, Git really focuses on these change sets in encapsulating a **change set as a discrete unit** and then those change sets can be exchanged between repositories. We're not trying to keep up-to-date with the latest version of something instead the question is, do we have a change set applied or not? So there is no single master repository, there is just many working copies each with their own combination of change sets. Let me give an illustration to make this clear.

Imagine that we have changes to a single document as sets A, B, C, D, E, and F, we're just going to give them arbitrary letter names so that we can help see it. We could have a first repository that has all six of those change sets in it. We can have repository 2 that only has four of those changes in it. It's not that it's behind repository 1 or that it needs to be brought up-to-date, it's just simply that it doesn't have the same change sets. We can have repository 3 that has A, B, C, and E, and repository 4 that has A, B, E, and F.

No one of these is right and no one of these is wrong. None of them is the master repository and the others are somehow out-of-date or out of sync with it. They all are just different repositories that happened to have different change sets in them. We could just as easily add change set G to repository 3, and then we could share it with repository 4 without ever having to go to any kind of central server at all. Whereas with CVS and SVN, for example, you would need to submit those changes to the central server and then people would need to pull down those changes to update their versions of the file.

Now by convention, we often do **designate a repository as being a master repository**, but that's not built-in to Git, it's not part of the Git architecture, that's just convention, that we say, okay, this is the master one and everyone is going to submit their changes to the master one, we're going to trying all stay in sync from that one or we don't have to. We can actually have three or four different master ones that have different versions, different features in them, and we could all be contributing to those equally and just swapping changes between them.

Now because it's distributed that has a couple of advantages, it means that **there is no need to communicate with a central server**, which makes **things faster**, and **it means that no network access is required to submit our changes, we can work on an airplane, for example.**

**And there is no single failure point**, with CVS and SVN if something goes wrong with that central repository that can be a real showstopper for everyone else who is working off of that central repository. With Git we don't have that problem, everyone can keep working, they've each got their own repository that they are working from, not just a copy that they're trying to keep in sync with a central repository. It also encourages participation in forking of projects, and this is really important in the open source community. Because developers can work independently, they can then make changes, bug fixes, feature improvements, and then submit those back to the project for either inclusion or rejection.

And if you decide you don't like the way that a open source project is going, you can fork it, take it to a completely different direction and say, you know what, I'm going to just make a clean break and make my repository now the one that I'm going to work from, all of my changes will be submitted to there, and I can still pull change sets from the master one into my project whenever I want. But I don't have to, I can go my own way. That becomes a really powerful and flexible feature that's well suited to collaboration between teams especially loose groups of distributed developers like you have in the open source world. Distributed version control is an important part of the Git architecture that you need to keep in mind.

Especially, if you have previous experience with another Version Control System like CVS or SVN. We'll talk a lot more about how Git tracks and merges these change sets as we go forward. For now just make sure that you understand that there is no central repository that we were from, all repositories are considered equal by Git, it's just a matter of whether a repository has change sets in it or doesn't.

**Who should use Git (A command line tool) ?**

1. Anyone wanting to track edits
2. Review a history log of changes made and view differences between versions
3. Retrieve old versions
4. Anyone needing to share changes with collaborators
5. Anyone not afraid of command line tools

