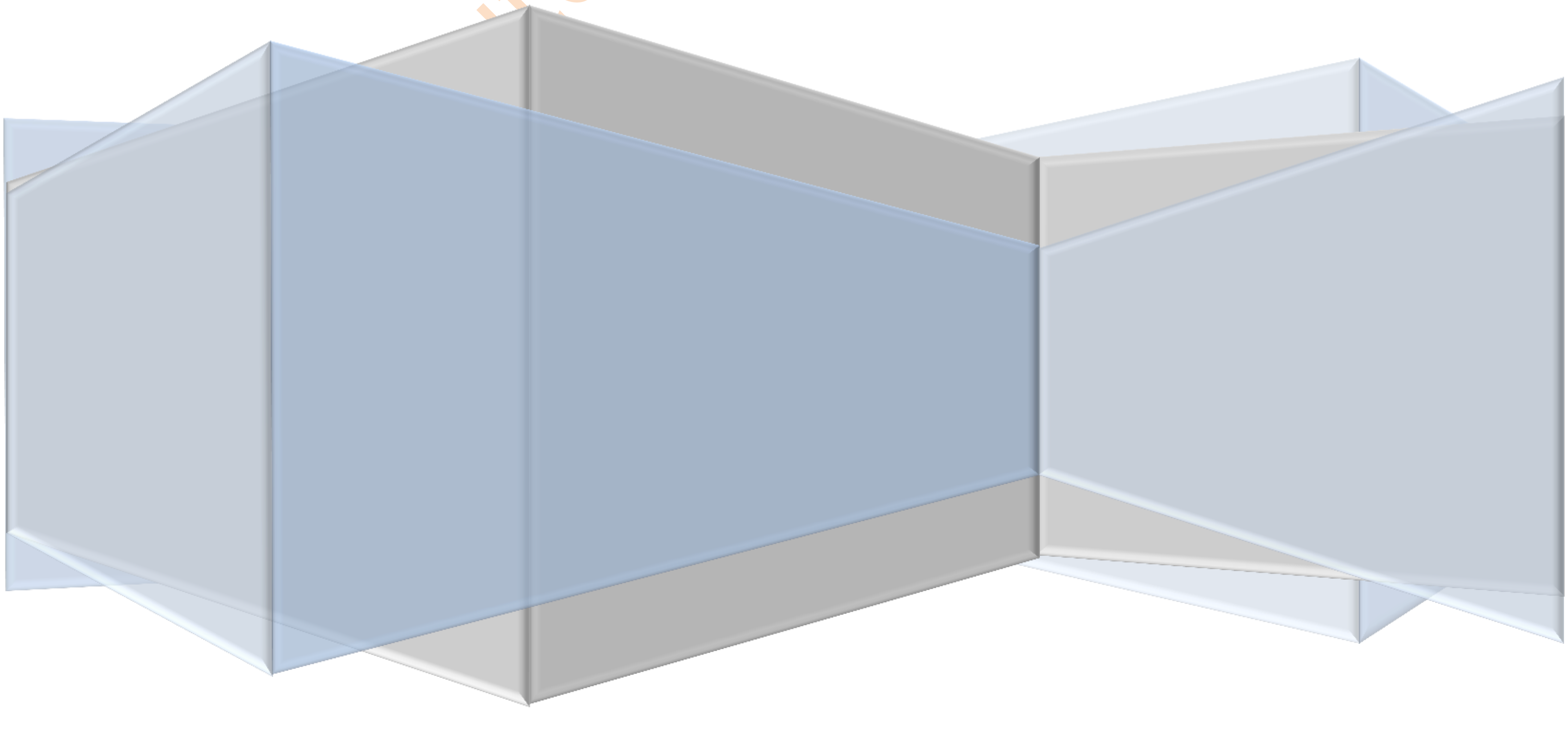
Automation Factory

GIT the Favourite DVCS

Ganesh Palnitkar



Version Control System:

Version control systems 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.

For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort.

Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

Software developers working in teams are continually writing new source code and changing existing source code. 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.

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use.

Version control software is an essential part of the every-day of the modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for non-software projects.

Benefits of version control

Developing software without using version control is risky, like not having backups. Version control can also enable developers to move faster and it allows software teams to preserve efficiency and agility as the team scales to include more developers.

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. One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Git is free and open source. The primary benefits you should expect from version control are as follows.

1. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software.

2. Branching and merging. Having team members' work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both.

While it is possible to develop software without using any version control, doing so subjects the project to a huge risk that no professional team would be advised to accept. So the question is not whether to use version control but which version control system to use.

What is Git?

By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

In addition to being distributed, Git has been designed with performance, security and flexibility in mind.

Performance

Unlike some version control software, Git is not fooled by the names of the files when determining what the storage and version history of the file tree should be, instead, Git focuses on the file content itself.

Every change to a file (content) is stored as a SHA1 code in the source code repository. This switching back and forth between file versions can easily be done with reference to the SHA1 code.

DVCS enables significant performance benefits as well.

GIT is tremendously fast. As compared to any other version control system, simple checking-in and checkout operations, even cloning entire repository action happens very fast.

Being able to perform and store entire repository actions locally, entire source code repository is always available even in case of no internet / network connection.

Security

Git has been designed with the integrity of managed source code as a top priority. The content of the files as well as the true relationships between files and directories, versions, tags and commits, all of these objects in the Git repository are secured with a cryptographically secure hashing algorithm called SHA1. This protects the code and the change history against both accidental and malicious change and ensures that the history is fully traceable.

Why GIT is famous

Git has the functionality, performance, security and flexibility that most teams and individual developers need. These attributes of Git are detailed above. In side-by-side comparisons with most other alternatives, many teams find that Git is very favourable.

Git is the most broadly adopted tool of its kind. This is makes Git attractive for the following reasons.

Tools like Bit Bucket helps to make use of GIT-HUB very easy to use and support.

Many third party software tools and services are already integrated with Git including IDEs.

If you are an inexperienced developer wanting to build up valuable skills in software development tools, when it comes to version control, Git should be on your list.

Git for developers

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.

A ‘Feature’ branch provide an isolated environment for every change to your codebase. When a developer wants to start working on something—no matter how big or small—they create a new branch. This ensures that the master branch always contains production-quality code.

Using feature branches is not only more reliable than directly editing production code, but it also provides organizational benefits.

Distributed Development

In CVCS, 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.

Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

Distributed development also makes it easier to scale your engineering team. If someone breaks the production branch in SVN, other developers can't check in their changes until it's fixed. With Git, this kind of blocking doesn't exist. Everybody can continue going about their business in their own local repositories.

And, similar to feature branches, distributed development creates a more reliable environment. Even if a developer deletes own repository, he can simply clone someone else's and start anew.

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.

Alternatively, junior developers can be confident that they aren't destroying the entire project by treating pull requests as a formal code review.

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.

In addition, Git is very popular among open source projects. This means it's easy to leverage 3rd-party libraries and encourage others to fork your own open source code.

Git works very well with continuous integration and continuous delivery environments. Git hooks allow you to run scripts when certain events occur inside of a repository, which lets you automate deployment to your heart's content. You can even build or deploy code from specific branches to different servers.

For example, you might want to configure Git to deploy the most recent commit from the develop branch to a test server whenever anyone merges a pull request into it. Combining this kind of build automation with peer review means you have the highest possible confidence in your code as it moves from development to staging to production.