Notes - Organizational Development - Version Control and Records

- Dr Nick Hayward

A brief introduction to version control and records with example usage for software projects.

Contents

Intro

Another key tool for software development is effective record keeping and management of versions for each code update added to a project.

The granularity of such records may vary depending upon context, from a single character for code revisions and versions to milestones for the overall project's development cycles.

Version control

Version control systems (VCS), in the current context of software development, are software tools that help developers manage changes to their source code over time. Such tools track modifications made to files, including when changes were made and by whom. This technology has revolutionised software development, making it easier for developers to collaborate on projects, maintain code quality, and improve productivity.

There are several version control systems available, each with its own set of advantages and drawbacks.

For example, specific options include the following

- local version control systems - these systems store changes locally in the files before being pushed to a single version of code in a database
- central version control systems - these systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place
- distributed version control systems - these systems, such as Git, allow multiple developers to work on the same codebase simultaneously

Examples of local VCS include

- Revision Control System (RCS) - one of the most common local VCS tools, uses a database to keep all changes to files under *revision control*. RCS maintains sets of patches, in effect the differences between the defined files. It may then collate all patches to recreate the state of a file at any recorded point in time.

A Concurrent Versions System customarily acts with a RCS to provide front-end options for operating on single files, expanding upon the basics of RCS to provide support for repository level tracking of changes. This option also adds support for a standard client-server model.

Examples include

- CVS

- Subversion - commonly referenced by its command line usage, SVN, and perceived as a successor of sorts to the earlier CVS. It maintains a central repository of the current and previous versions of files and directories.
- Perforce - another centralised version control system, often deployed in enterprise, provides various features such as *atomic* commits and good support for handling binary files
- …

Common examples of distributed VCS include

- Git - by most metrics, the most popular distributed version control system, known for its speed, workflow compatibility, and open-source foundation, &c.
- Mercurial - similar to Git, often noted for its inherent simplicity and adaptability to projects of varying size
- others include
  - GNU Bazaar
  - …

and common examples of hosted VCS include

- GitHub - https://github.com/
- GitLab - https://about.gitlab.com/
- Bitbucket - https://bitbucket.org/
- …

Suffice to say, there are many different options for version control systems, commonly chosen and deployed relative to specific domain or project requirements. For example, as noted above, there are options beyond simply defaulting to Git usage for a given project.

Git version control

Git is the most popular distributed VCS and has gained widespread adoption among developers and companies worldwide. It allows developers to track changes made to their codebase, collaborate with team members, and revert to previous versions if needed.

Git is open-source and can be used on various operating systems, making it a popular choice for individual developers and large corporations. Many companies, like Google and Facebook, use customised versions of Git to manage their massive codebases.

Specific examples of Git usage for projects include the following

- Linux kernel
- Django
- Ruby on Rails
- Visual Studio Code
- Android
- Node.js
- and many more…

Benefits include the following

- faster releases - Git's efficient handling of branches and merges may allow for quicker and more frequent releases
- simultaneous development - Git's distributed nature allows multiple developers to work on the same project simultaneously
- strong community support - Git has a large and active community of users providing potential support and contributions to its ongoing development
- built-in integration - Git has been integrated with many popular development tools and platforms
- offline working - developers may work offline with Git with a complete copy of a project's history in their local repository

- Branch workflow - a noted advantage of Git is its branching capabilities, which, compared with centralised version control systems, creates branches that are cheap and easy to merge
- distributed development - each developer gets their own local repository, complete with a full history of commits
  - i.e. a full local history makes Git fast
  - a user does not require a network connection to create commits, inspect previous versions of a file, or perform diffs between commits &c.
- pull / merge requests - many source code management tools such as GitHub, GitLab, Bitbucket &c. provide support for pull requests
  - i.e. a pull request is an option to ask another developer to merge one of your branches into their repository
- …

and some noted issues and problems may include the following

- complexity - Git's distributed nature can make it more complex to use, e.g. compared to centralised version control systems. This complexity can be a barrier to entry for new contributors, especially those who are not familiar with version control systems.
- lack of universal GUI - whilst GUI tools are available for Git, in various forms, they often fail to expose all of Git's features. This means some users may require access to, and familiarity with, a system's command line, thereby creating a further barrier to entry.
- handling large files - Git can struggle with large files or large repositories. This can slow down operations and make the repository difficult to manage.
- merge conflicts - whilst Git's branching and merging features are powerful, they may also create complex merge conflicts. Resolving such conflicts may be time-consuming, and inherently error-prone.
- lack of access control - by default, a cloned repository provides access, at that context level, to a project's codebase and its history. This lack of access control may be a concern for various projects, which may need to limit parts of the code to specific roles &c.
- …

In spite of these potential challenges and perceived limitations, many open-source projects find that the benefits of using Git outweigh the drawbacks.

Versioning workflow

We might also consider various versioning patterns, commonly referenced for project development and management.

For example, *Gitflow* and *GitHub Flow* are branching models used in software development, specifically with Git. Each option provides guidelines on branch organisation and managing code changes in a collaborative environment.

- Gitflow - defines a methodology to help with continuous development, enhancement, and bug fixes, including branches for *feature*, *develop*, *release*, *hotfix*, and *master/main*.
- GitHub flow - a simple workflow pattern, emphasising continuous deployment, which defines use of a single branch in a repository for ongoing work, such as documentation, outlines, &c.

Atomic commits

As noted aboved, one of the benefits of version control systems is the option to effect *atomic* commits as a specific option for various projects and contexts.

An atomic commit is an operation, which applies a set of distinct changes as a single operation. If the changes are applied, then the atomic commit is recorded and marked as a success. If there is a failure before the atomic commit may be completed, all of the completed changes in the atomic commit are reversed. This ensures that the system remains in a consistent state.

In this context, the term *atomic* means indivisible. In effect, an atomic commit may be considered a single, indivisible unit of work. As such, all changes made in an atomic commit may be considered contiguous, and may be committed to the repository as a single unit.

Atomic commits are essential for multi-step updates to data. They make code reviews faster, and reverting to an earlier state easier and safer, since they can be applied or reverted without any unintended side effects.

For example, if a developer makes a feature change, and then adds an unrelated bug fix, it becomes more complex to revert to a specific earlier state. If the feature change, for example, is subsequently deemed slow or unpopular, it may be necessary to roll back the update. If the earlier commit only included the feature change, reverting to the earlier state would a simple matter of rolling back the commit. However, the earlier combination with the *bug fix* introduces complicatons making the reversion of the commit more complex.

Alternative versioning compared to Git

We may also briefly consider how Git compares with alternative versioning systems such as CVS, SVN, and Mercurial for software development.

For example,

- Git vs. CVS
  - CVS (Concurrent Versions System) is an older, centralised version control system. Git, as noted, is a distributed version control system.
  - Git offers atomic operations, meaning all operations are either fully completed or not done at all, ensuring the repository is always left in a consistent state. CVS, being a set of scripts around the per-file RCS version control system, does not offer atomic operations.
  - Git stores changes to the entire project, while CVS changes are per file. This makes it easier in Git to revert or undo whole changes.
  - Git has a more complex setup process compared to CVS. However, Git's distributed nature allows for better local version control.
- Git vs. SVN
  - SVN (Subversion) is a centralised version control system. Git is distributed, allowing developers to work on their own local copy of the entire project.
  - Git provides an easy-to-use version control system that is distributed and has a strong branching system that can help with source code management. SVN, by contrast, has a good access control system and a learning curve similar to that of Git.
  - Git is more complex, with more commands, and requires your team to know it inside and out before using it safely and effectively. SVN is simpler and more straightforward for new users.
- Git vs. Mercurial
  - Mercurial, like Git, is a distributed version control system. However, Mercurial is often noted for its simplicity.
  - Git offers more granular control at the cost of requiring a little more user input, while Mercurial makes its default behavior what most users want ~80% of the time, at the cost of some granularity of control.
  - Git is appropriate for projects that call for sophisticated branching and merging, accepted industry standards, and a sizable development community. Mercurial is a suitable option for teams with less technical know-how, tasks that value simplicity and usability, and projects that call for a scalable version control system.

Why choose a version control system other than Git for specific projects and development? Various project or domain specific reasons may be considered, but the following are some common reasons.

For example,

- simplicity - some alternatives to Git, like Mercurial, are noted for their simplicity. They might be more suitable for teams with less technical know-how.
- centralised version control - some teams may prefer a centralised model, including options such as SVN or Perforce. In these systems, all changes are made directly on a central repository, which can be

simpler to manage in certain contexts.
- large files - Git can struggle with large files or large repositories. If a project involves large files or has a long change history, alternatives like Perforce might be more suitable.
- locking mechanisms - some version control systems offer a *locking* mechanism, where important files, particularly large binaries, may be *locked* to prevent other developers from changing these files until permitted. This feature can be useful in certain development environments.
- integration with existing tools - some teams might choose a specific version control system because it integrates well with other tools they're already using for a given project
- legacy code - in some cases, teams continue to use older version control systems because they have legacy codebases managed in these systems, and migrating to Git might be costly or potentially error prone
- …

In summary, the choice between Git, CVS, SVN, and Mercurial depends on the specific needs of the project and the team's familiarity with the tool.

Summary of benefits of version control

As we consider various options for version control systems, and their potential usage and impact on a project's ongoing development, we might summarise their benefits as follows.

For example,

- collaboration - version control systems, in particular distributed examples such as Git, allow multiple developers to work simultaneously on the same project. They help teams work faster &c., particularly in DevOps teams.
- tracking and managing changes - version control systems keep track of every modification to the code. If a mistake is made, developers may revert changes and compare earlier versions of the code. For example, to help fix a mistake whilst minimising disruption to all team members.
- protecting source code - for almost all software projects, the source code is a primary and important asset. Version control, when used correctly, protects source code from potentially catastrophic errors, perhaps due to human error or simply unintended consequences.
- reducing conflicts - changes made in one part of a project's software can be incompatible with changes and updates introduced by another developer working concurrently. Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.
- testing and development - until tested, and subsequently verified for quality, updates to code may introduce new bugs and issues. Version control helps promote testing and development, which may proceed in unison until a new version is ready.
- workflow flexibility - good version control software should help support a developer's preferred workflow without imposing a particular way of working. Ideally, such software should work on any platform, rather than dictate what operating system or tool chain developers must use.
- efficiency and agility - version control helps developers adapt, pivoting to new options and solutions where necessary, whilst also allowing software teams to preserve efficiency and agility as the team scales to include more developers.

If we consider such benefits, we may see how crucial and essential version control systems have become to modern software development and code-based projects.

Version control and DevOps

If we consider the broad topic of *DevOps* for a moment, we might review some of its important contributions to modern software development and project management.

For example,

- enhanced collaboration - DevOps brings together development and operations teams, reducing silos and fostering a culture of shared responsibility and transparency, leading to improved collaboration,

faster feedback loops, and increased efficiency
- faster releases - through increased efficiencies, improved team collaboration, automation tools, and continuous deployment, teams can reduce the time from product inception to market launch
- adaptability - a DevOps culture encourages teams to have a customer-first focus. By combining agility, team collaboration, and focus on the customer experience, teams may continuously deliver value to their customers and increase their competitiveness in a given market.
- system stability and reliability - adopting practices of continuous improvement, teams may develop increased stability and reliability of the deployed products and services. These practices help reduce failures and risk.
- improved recovery time - *mean time to recovery* metric indicates how long it takes to recover from a failure or breach. To manage software failures, security breaches, and continuous improvement plans, teams should measure and work to improve this metric.
- cultural shift - to successfully implement DevOps, teams need to be embrace a DevOps culture. Cultivating this culture may require various structural and cultural changes in the way people work and collaborate. When organisations commit to a DevOps culture, they may create and foster an environment to encourage the evolution of high-performing teams.

In effect, *DevOps* may be considered as more than simply a set of defined tools and practices. It might also be considered akin to a suggested methodology, promoting collaboration, efficiency, and continuous improvement. For software developers, it suggests many discernible benefits, including enhanced collaboration, faster development cycles, improved software quality, and a diversification of developer skills for the current project team.

If we then consider the role of version control relative to DevOps, we may see some of the following benefits and, indeed, how they overlap to provide a reliable and practical workflow for improved software development and project management.

For example,

- enhanced collaboration - version control systems, in particular distributed examples such as Git, enable multiple developers to simultaneously work on the same project. They help teams work faster, employing, hopefully, safer development practices, particularly in DevOps teams.
- tracking and managing changes - version control systems keep track of each code modification. If an error or bug is created, developers may revert to a different version, compare and contrast earlier versions of the code, helping to fix a mistake whilst minimising disruption to team members.
- protecting source code - for software projects, their source code is a key resource and asset. Version control inherently provides safeguards and options to protect source code from unintended, accidental, or simple degradation of quality due to human error and abuse.
- reducing conflicts - changes added to one part of the software may be incompatible with others introduced concurrently by another developer. Version control helps teams resolve such potential conflicts, tracking individual changes by each contributor and helping prevent concurrent work from conflicting.
- testing and development - software development, and code changes, may introduce new bugs, which can't be trusted without further testing and validation. Testing and development should proceed in unison until a new version is ready.
- workflow flexibility - good version control software supports a developer's preferred workflow without imposing a restrictive workflow. Ideally, it should also work on any platform, rather than stipulate a specific required operating system or tool chain for developers to adopt.
- efficiency and agility - version control also helps developers adapt efficiently, allowing software teams to preserve efficiency and agility as the team scales to include additional developers

Such defined benefits clearly show how version control is a key part of modern software development and, in this context, DevOps for projects and their associated teams.

Resources

- 4 Key Benefits of DevOps - Atlassian - https://www.atlassian.com/devops/what-is-devops/benefits-of-devops
- About version control - Git SCM - https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- Comparison of version-control software - Wikipedia - https://en.wikipedia.org/wiki/Comparison_of_version-control_software
- Gitflow workflow - Atlassian - https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
- GitHub flow - GitHub Docs - https://docs.github.com/en/get-started/using-github/github-flow
- Should you use a Git alternative - How-to-Geek - https://www.howtogeek.com/devops/should-you-use-a-git-alternative/
- Version Control Systems - Geeks for Geeks - https://www.geeksforgeeks.org/version-control-systems/
- What is DevOps? - Microsoft Learn - https://learn.microsoft.com/en-us/devops/what-is-devops
- What is version control? - Atlassian - https://www.atlassian.com/git/tutorials/what-is-version-control