A single source of truth

Version control; what every embedded software engineer needs to know about Git and GitHub. By **Sven Erik Knop**.

Version control – increasingly a *de facto* part of the embedded software development process – has changed a lot in the past few years. This has major implications for any organisation reviewing its version control strategies and making investments in versioning tools.

One of the biggest developments has been the popularity of Git, an open-source version control system originally developed in the Linux world, but now in widespread use. Pure Git is designed for independent working and distributed localised versioning, with no central server functionality. Git's appeal is understandable because it offers individual developers some compelling benefits – it is cheap, fast to set up and gives individuals or small-to-medium teams the rapid, flexible versioning control and workflows they want.

While many traditional embedded software developers may not be embracing Git, the new generation of engineers coming out of university is – as well as new development methodologies such as Agile.

Git does have some limitations, however, and these can be overcome to some extent. More on that later, but first, GitHub deserves an explanation, because it is often referenced in the same conversations as Git. While Git is a version control tool, GitHub is a Git management tool, giving users a common place to store all the Git repositories that are created and, in theory, making it easier to manage them. GitHub is responsible for popularising Git, but it's certainly not the only Git management tool around, with others each having their different features and approaches.

Parallel developments

At the same time as the rise of Git, the very nature of software development has changed. Projects have become larger and more complex, often involving bigger teams dispersed across the world. Instead of just versioning source code, users often want to include binary files to keep everything associated with a project in one place. This has been further accelerated with the Internet of Things. Multigigabit repositories are now commonplace, even in small organisations, and this puts pressure on companies to ensure that digital IP is safe, accessible and auditable, but without slowing down the development process. There is also more focus on the entire lifecycle of digital assets and not just their initial development. These drivers mean that Git's role needs some careful consideration.

Security

The IoT is at risk of taking us back 30 years to the time when network stacks were being built without inherent security measures in place, therefore risking, for instance, massive DDOS attacks. Software development of all kinds needs more rigour and due diligence around security; not only to meet compliance and regulation, but also safety requirements. Git's architecture does not include security beyond a verified history (no authentication, no permissions); but that's not a criticism as it's not designed to be a security tool.

Git management can be used to bring in security, but the nature and level of that security varies. Some tools provide access to files at the repository level, so security is all-or-nothing: either users can access all files in the repository, or none at all. Other tools provide access control at both the repository and branch level. More advanced tools provide a finer-grained level of security. For instance, Perforce Helix provides access control (and visibility of anyone trying to access) across IP address, user and group, with enforceability at code repository, branch, directory or individual file level, locally or across authorised locations.

Development at scale

Git is fine for many smaller projects and teams, but it quickly becomes unwieldly when projects grow. This is due to its architecture: a Git repository is organised internally as a tree where each node represents a change to a set of files, cryptographically secured via a hash to its previous change. Each repository requires its whole history to be present, so in a big project, downloading ('cloning') a repository can slow from seconds to minutes or even hours.

Making the project smaller is not an option: users are stuck with the Git repository configuration they originally specified; it cannot be split up. Having lots of repositories is one approach tried by some organisations, but as they have found, it is very hard to manage hundreds of Git repositories – hence the term 'Git sprawl'. There is no single view of the whole project anymore. While it's possible to 'stitch' all those repositories back together again, that introduces a lot of extra work. In organisations that have widespread Git adoption, it's not unusual to employ teams just to manage Git, so the initial low-cost of Git may be outweighed by the further investments needed to make it viable.

Git management tools help, because they can – to a large extent – add ways to make Git more manageable in large environments. For instance, Perforce Helix gives users the flexibility to break a project into smaller chunks retrospectively. Git management tools are proving popular and are certainly solving some real-world problems. Equally, many organisations which know they cannot avoid complex, large-scale development environments, are choosing to avoid Git as much as possible, although it is difficult to mandate to developers that they cannot use Git.

The 'single source of truth'

In embedded development, organisations typically want to include everything associated with a project as a 'single source of truth', with all digital assets in one place, completely traceable and visible. It is increasingly viewed as a fundamental requirement of DevOps and supports compliance processes by providing a single, immutable history of a project.

In the embedded market, development environments typically include a wide variety of binary files, often large and in different formats. The problem is that binary files are not easy to version in Git – even the use of GitLFS will lead to large volumes of asset updates – so companies may end up storing binary assets in other systems, thus undermining the 'single source of truth'. Also, projects may include staff outside software development teams who do not find Git easy to use.

"Git is fine for many smaller projects and teams, but it quickly becomes unwieldly when projects grow."
Sven Erik Knop

A hybrid solution

While Git is not going away soon, neither are the problems associated with complex, large-scale development projects. There are a variety of options, including hybrid systems that support both Git and other version control environments (such as Perforce Helix), both distributed and centralised version control. Techniques such as 'narrow cloning' also help by breaking sprawling source code into smaller chunks, but again, that's extra work that an organisation has to factor into its workload and budget.

The overall message is this: Git has a lot to offer and it has played a great role in bringing version control to new users. However, its enterprise-grade deployment needs careful consideration and management when projects begin to scale or become more complex, particularly around cost, security, speed and flexibility. Fortunately, there is an increasing number of ways to achieve that and those will continue to evolve, giving companies more choice.

Author profile:
Sven Erik Knop is technical marketing manager with Perforce Software.