PERFORCE

SEVEN TIPS FOR SUCCESS WITH GIT IN THE ENTERPRISE

Introduction

Git's popularity for open-source and other small- to medium-size projects has led inexorably to the enterprise, but making Git work in that arena is neither easy nor obvious. In fact, its adoption creates a number of challenges to balance developer preference with the needs of the enterprise. This guide reviews considerations when using Git in the enterprise and recommends best practices for Git deployment.

1 WORKFLOW

The most obvious reason for Git adoption is because developers like its distributed workflow, even though it's rarely used in an entirely distributed model. They often need to switch contexts frequently, and Git's support for lightweight, in-place, local branching is a big help. In fact, Git's branching may be its defining feature.

But once you move beyond the individual desktop, how do you manage the new complexity introduced by Git? Who gets to determine the final release branching structure? What is the protocol for sharing experimental branches with colleagues? Where should the work be reviewed? When should those branches' contents be brought back into the master? And how should that be done, given that the Git community remains polarized over merge versus rebase?

One approach is provided by the **Git-flow** proposal, although it's not an ideal solution for all teams. Tools like **GitSwarm**, **GitHub**, **GitLab**, or **Atlassian Stash** implement simple workflows around pull requests to supply some of the answers.

The best time to address these issues is before teams start adopting Git, because trying to change workflows and rebuild history once work has been committed is a very messy business.

WORKFLOW BEST PRACTICES

- Nail down your branching strategy with clear stages and triggers, preferably automating the flow of code as much as possible.
- Make review an integral part of that branching strategy; multi-disciplinary collaboration is crucial to modern product development.
- Give all of your stakeholders tools suited to their particular needs to maximize productivity and ease of use.

2 CONTENT

A popular dictum of Agile development is that you should keep all of your intellectual property (IP) in the same place, a "single source of truth." Yet increasingly, multi-disciplinary teams and requirements have exploded the sheer number and size of assets for a typical project. Source code is often a tiny drop in the binary bucket compared to documents, images, models, audio, video, even entire virtual-machine (VM) environments for the sake of testing and deployment.

This expansion poses a serious challenge for enterprise Git adoption because the design of its internal file system mandates a practical maximum repository size of a gigabyte or two at most. Even repositories far short of the practical limits can exhibit relatively poor performance, depending upon the type and size of assets and operation at hand; executing a simple Git "blame" command against a repository with large digital assets can provide a painful demonstration of the point.

In addition, these binary assets are being developed by product designers or artists who may lack technical skills and may be unlikely to use the Git command line, even if it could handle their work. Part of your Git management plans should include how these non-technical contributors will store their work, how those files will be managed alongside the code in Git, and how all the correct revisions of files will be brought together in your build and release systems.

The most popular ways to handle large binary assets are to move large files outside the repository or to divide the content among multiple repositories, which must be unified through DevOps magic for builds, testing, releases, and other tasks. Tools such as **git-annex** and **Git LFS** can help in moving digital assets outside the repository while leveraging **Git submodules**, and a variety of home-brewed scripting techniques can be useful in taming "Git sprawl".[1] However, these approaches can introduce new challenges when considering backup processes or distributing content across different locations. Where possible, companies should look for a version control system that provides the benefits of a distributed version control system (DVCS) but can keep all the assets in a single store.

CONTENT BEST PRACTICES

- Keep all of your content in one place, ideally in a single version management tool. Failing that and if you choose to use Git, then consider adopting a "monorepo",[2] instead of settling for tools that fragment your IP.
- Use a Git management suite that can handle all of your files, not just source code but also digital assets.
- Have a strategy for archiving older content that dovetails with your branching strategy to keep working repositories clean and easier to use.

1. Git sprawl refers to content divided into multiple repositories to keep size down and performance up.
2. A "monorepo" is a single large repository. It can make management easier in some respects but one must also consider the limitations of large repositories in Git.

3 CONTINUOUS DELIVERY

Whatever workflows your contributors favor or type of content they create, the final products have to be integrated and tested with as much automation as possible. The architecture of Git can make this automation a heavy load on build systems, so planning is necessary to get reasonable performance from continuous delivery systems.

A common best practice for continuous delivery is to run builds and tests against a fresh copy of all your files. But in the enterprise, this step can easily overload servers when using Git. There are two reasons for this. First, like most DVCS systems, Git brings down all revisions by default during a clone operation. And second, Git has no support for narrow cloning (bringing down only the necessary files instead of everything). In short, when you clone with Git you get every revision of every file by default.

These issues typically aren't problems for open-source and other small- to medium-size projects, but they can become fatal obstacles for larger enterprise projects. The sheer volume of merge operations when dealing with a large number of development branches can create a similar stumbling block. Git encourages developers to branch for each task or feature, but keeping all that work in sync in both directions can be tricky with large teams working with lots of files.

CONTINUOUS DELIVERY BEST PRACTICES

- Use shallow clones to get just head revisions, and prefer a Git management solution with support for narrow clones; both will greatly ease server load.
- Choose the right build cycle times/events for your projects; building every hour or day (instead of on every commit) may be necessary when using Git.
- Automate code merging as much and as frequently as possible to catch integration issues before they log-jam the flow of code.

4 RELIABILITY

No matter how well a tool does a job, it's not useful when it's not working. This is especially true in the enterprise, which often runs around the clock around the world. Git was built largely with the simple file system in mind, and most Git management solutions are built directly on top of basic Git. So meeting enterprise requirements for disaster recovery and high availability can be challenging.

Most Git management solutions offer services such as backups and snapshots, which are a good first line of defense. Some offer higher availability through standby VMs with various means to mirror changes between file systems and/or swap out different file storage as needed. But if what you need is dial-a-load scalability with high availability, your list of options narrows considerably.

Although it may appear to be simple to clone another copy of a repository for safety, you'll be cloning the whole repository with Git. This approach is fine for small or even medium-size projects, but big projects consume bandwidth, disks, and time. If your content is measured in anything beyond megabytes, you need to dig carefully into the details of reliability with your Git management solution.

RELIABILITY BEST PRACTICES

- Backups are good, but a disaster recovery plan is even better; craft one to balance acceptable loss with cost and verify it regularly.
- When selecting a Git management solution, look for one that offers both clustering and high availability to keep everyone working even when hardware goes down.

5 SECURITY

Raw Git has a limited approach to security, providing solid authentication while ignoring authorization altogether. That is, it cares about who you are but leaves what you do to the file system. This approach is great if all you want to do is ensure each commit author is who he or she claims to be because identity is verified through the use of public-key cryptography signatures.

But what if you want to restrict access to a particular repository? A particular branch? How about a single file full of proprietary secrets? Git offers nothing to deal with these needs, which is a key reason so many Git management solutions exist. But not all Git management solutions are created equal when it comes to security.

SECURITY BEST PRACTICES

- Be sure the branching structure you choose not only matches your workflow but also carves up your content according to security needs.
- Select a Git management solution that supplies access control to the maximum necessary granularity for your security needs.
- When branches aren't enough, organize your content ahead of time so that critical files are together to simplify restricted access.
- Choose a Git management solution with active and timely monitoring of user actions to flag and report questionable behavior before you lose IP.

Git management solutions for the enterprise typically make it easy to create users and groups and to restrict project (read: repo) access using those tools. In addition, there are typically a small set of roles that have varying permissions that you can assign. Some (e.g., **GitLab**) extend the notion of permissions to branches as well, making it possible to restrict access to some users but not others.

If your needs extend beyond that (e.g., finely grained permissions at the folder/file level), a solution is much harder to find. The best solutions will offer various levels of monitoring and track patterns of use.

Consider also your needs for audit trails and logging. In a secure or regulated environment, you may be required to keep immutable logs of who changed what, when, and why. Standard Git allows a lot of flexibility when it comes to rewriting history or even hiding changes completely. If audit trails are important, look for a version management tool that provides reliable, secure logging.

6 REPOSITORY MANAGEMENT

The need for repository management varies greatly depending upon the limitations of your chosen Git management solution. If you choose a system that poorly handles digital assets, large numbers of files, or a large total repository size, then Git sprawl may be your new way of life.

REPOSITORY MANAGEMENT BEST PRACTICES

- Consider how much bandwidth your DevOps team has for repository management when reviewing your content and choosing a solution.
- Investigate tools and techniques for handling components at a higher level as first-order objects in themselves if at all possible.
- Use a Git management solution that avoids Git sprawl as much as possible or at least takes care of the details for you under the hood.

Similarly, you may also incur Git sprawl if your process relies heavily on component-based development (CBD). Large numbers of individually developed and versioned components can lead to issues with **Git "submodules"** if your Git management solution doesn't provide another mechanism. Git by itself makes it relatively easy to map external repositories into another, but there are a number of different tools, approaches, and trade-offs to consider.

Regardless of the cause, the more finely you have to break apart your content, the more problems you'll have trying to put it back together for builds, testing, and other tasks. Managing the explosion of repositories can become a full-time DevOps headache without careful planning and maintenance.

7 VISIBILITY

The last stop on our tour of enterprise needs is another that Git by itself ignores entirely and most Git management solutions do little (if anything) to address: the need for transparent visibility into the production pipeline at every stage. Most Git management solutions give you dashboards for all projects and individual projects and even some handy statistics. Many also provide lightweight issue tracking or the ability to integrate with third-party application lifecycle management (ALM) tools.

But unless you go with a vendor that supplies other parts of the total solution as well, you're likely to be on your own handcrafting plugins, scripts, or other integration mechanisms. The aforementioned Git sprawl only complicates matters—for obvious reasons, it's much easier to look at data from a monorepo than data from hundreds or even thousands of repos.

VISIBILITY BEST PRACTICES

- Assess your information priorities carefully and determine the most crucial metrics; let them guide your Git management solution selection.
- Be mindful of the degree to which the solution you choose will lead to Git sprawl; investigate tools and techniques for aggregating data if needed.
- Select a vendor that supplies more than just Git management, particularly if you require custom metrics or broad integration with other tools.

CONCLUSION

We've looked at seven key aspects in which Git poses a challenge for the enterprise. The best option for your enterprise may be to find an alternative to Git that addresses the company's needs while still delivering the features that developers love.

There are few such tools currently available, and a leading solution is **Perforce Helix**. If you do choose to deploy Git, the best practices listed in this ebook will help you to identify important planning considerations, select a Git management solution, and minimize issues in the long term. So take a careful inventory of your processes, content, and needs before diving into the shallow end of the wrong pool with Git in the enterprise.

North America
Perforce Software Inc.
2320 Blanding Ave
Alameda, CA 94501
USA
Phone: +1 510.864.7400
info@perforce.com

Europe
Perforce Software UK Ltd.
West Forest Gate
Wellington Road
Wokingham
Berkshire RG40 2AT
UK
Phone: +44 (0) 1189 771020
uk@perforce.com

Australia
Perforce Software Pty. Ltd.
Suite 3, Level 10
221 Miller Street
North Sydney
NSW 2060
AUSTRALIA
Phone: +61 (0)2 8912-4600
au@perforce.com

PERFORCE

perforce.com