Paper PP18

Tailoring Git for SAS Projects: Overcoming Challenges, Achieving Excellence

Rajwanur Rahman, Phastar, Bangladesh

ABSTRACT

Implementing version control in SAS projects presents unique challenges compared to other development projects, like web. However, our organization has successfully adopted Git, a powerful distributed version control system, by embracing its flexibility and adapting it to the specific requirements of SAS projects. Through an innovative approach, we have tailored our workflow to leverage Git's capabilities, ensuring regulatory compliance, validation, and an auditable trail of changes. Git's branching and merging features have revolutionized our code review processes, fostering seamless collaboration among teams. Furthermore, the ability to track commit history has significantly improved productivity by enabling efficient issue identification and resolution. By overcoming the challenges of adapting Git to SAS projects, we have achieved excellence in project management, delivering high-quality, reproducible SAS solutions to our pharmaceutical clients.

INTRODUCTION

Git has become the industry standard for version control across a wide range of software development projects. However, integrating Git into SAS programming workflows presents unique challenges. Unlike web development, where Git integration is often seamless, applying it to the SAS environment requires a more nuanced approach. This paper outlines our organization's journey in adapting Git for SAS projects, highlighting the obstacles encountered, the solutions developed, and the significant benefits achieved. We share our experiences, from initial skepticism to widespread adoption, demonstrating how Git has transformed our SAS projects.

BACKGROUND AND CHALLENGES

THE STARTING POINT: A FAMILIAR SCENE

Prior to implementing Git, our SAS programming workflow mirrored the practices common across the industry. SAS programs resided on shared network drives, with version control managed through manual file naming conventions (e.g., program_v1.sas, program_v2.sas). Tracking changes and identifying the authors of specific modifications proved cumbersome and inefficient. Furthermore, concurrent programming was fraught with the risk of accidental overwrites, creating an environment of uncertainty and potential data loss. The absence of a structured system for recording changes made auditing and reconstructing the development history a complex and error-prone process.

THE REAL CHALLENGES: OBSTACLES TO ADOPTION

Introducing Git into this established workflow presented several key challenges. Initially, there was resistance from the apparent complexity of Git. Many SAS programmers viewed Git as an mysterious and overwhelming tool. Team leads expressed concerns about the additional overhead of managing Git-specific tasks and the potential disruption to existing workflows. The prevailing sentiment was often "why fix what isn't broken?".

Beyond these initial perceptions, we also faced significant quality control and resource allocation challenges. Maintaining our stringent validation processes within a Git framework required careful consideration. Questions arose about managing development and production outputs, ensuring traceability, and meeting audit requirements. Moreover, implementing Git required investment in training and infrastructure changes, placing additional demands on project resources.

PILOTING THE SOLUTION: A PRACTICAL APPROACH

STARTING SMALL: THE PILOT PROJECT

To address these challenges and demonstrate the feasibility of Git integration, we initiated a pilot project. We selected a mid-sized clinical study with a team of three programmers, representing varying levels of prior version control experience. One programmer had experience with Git, another with SVN, and the third had no prior experience with either system. This diversity allowed us to assess the learning curve and identify potential training needs.

The two-month pilot provided invaluable insights. We learned what worked well, where training was needed, and which processes required simplification. This iterative approach allowed us to refine our strategy and tailor Git to the specific demands of our SAS environment.

INFRASTRUCTURE THAT WORKED: A SECURE AND INTEGRATED SYSTEM

We established a practical and secure infrastructure for our Git implementation, leveraging SAS Studio:

- **Self-hosted GitLab:** We opted for a self-hosted GitLab instance behind our firewall, providing enhanced control and security over our codebase.
- **File Server Integration:** The GitLab instance was integrated with our existing file server, ensuring seamless access to project files and minimizing disruption to established workflows.
- **Single Sign-On:** We implemented single sign-on (SSO) for streamlined access and simplified user and repository management.
- **Dedicated User Areas:** We established separate user areas, providing isolated workspaces for each programmer. This approach allows users to work independently on their projects without interfering with other users or the main project area, obsoleting the risk of accidental overwrites and promoting parallel development. These user areas are then linked to the Git repository for version control and collaboration.

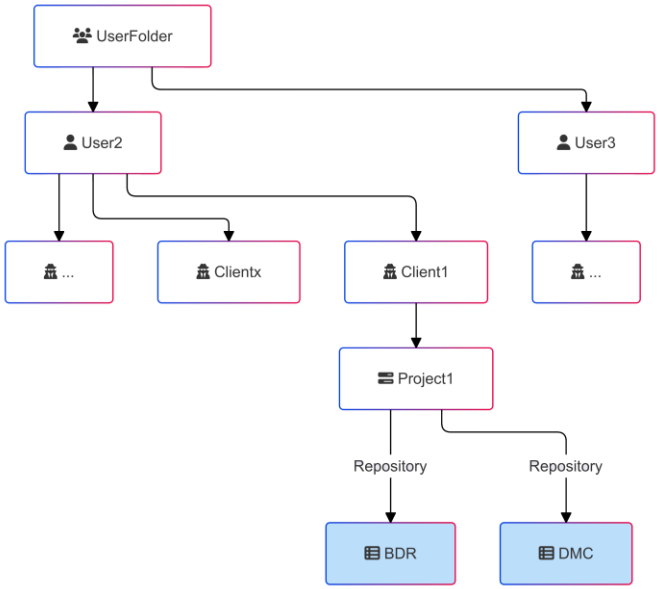

Figure 1: User area folder structure

THE HEART OF THE SOLUTION: THE INIT_GIT_ENV MACRO

The cornerstone of our successful Git integration lies in the development of a custom SAS macro, init_git_env. This macro serves as a bridge between the complexities of Git and the familiar SAS programming environment. By encapsulating complex Git operations behind a simplified interface, init_git_env makes Git accessible to SAS programmers without requiring them to become Git experts. This abstraction layer empowers our team to leverage the power of Git for version control while minimizing the learning curve and reducing the potential for errors.

The init_git_env macro automates a range of essential Git operations, streamlining the workflow and enhancing productivity. This macro automatically determines the necessary configurations to clone existing repositories or pull updates from repository server. It seamlessly handles branch switching, allowing programmers to navigate between different versions of the codebase with ease. Furthermore, this macro facilitates pulling changes from other branches and essentially merging with the current one, ensuring that local copies are kept up to date with the latest developments.

Beyond core Git operations, init_git_env integrates seamlessly with our existing SAS infrastructure. It works in concert with our established bootloader macro, maintaining consistent folder structures and preserving familiar workflows. This integration minimizes disruption to existing processes and allows for a smooth transition to the Git-based version control system. The macro also intelligently manages user areas and permissions, ensuring that programmers have the appropriate access levels while safeguarding sensitive data and code. Built-in validation checks prevent common Git-related errors, providing clear and informative error messages to guide users. A dedicated debug mode further aids in troubleshooting during development and implementation.

Example Usage:

```
/* Basic usage leveraging bootloader variables */
%init_git_env;

/* Advanced usage with explicit parameters */
%init_git_env(
    client = ABC,
    project = XYZ,
    reporting_effort = BDR,
    userarea = S:\SasUsers,
    branch = UserName1234,
    debug = Y
);
```

A STREAMLINED WORKFLOW

Our streamlined Git workflow begins with repository cloning. Programmers initiate their work by cloning the project repository into their designated work areas on the file server. This process is simplified by the git_init_env macro, which streamlines cloning and other essential Git operations within the SAS environment. The macro automatically sets the current branch to a user-named branch, ensuring isolated development, and pulls the latest changes from that branch. It also offers the option to pull updates from other branches, such as the main branch, provided no conflicts exist.

Following the initial setup, programmers develop and commit their changes. They work on SAS programs and project documents within SAS Studio or using external editors. Changes are committed locally, either through SAS Studio or a Git GUI tool provided within our development environment. Crucially, only SAS programs and designated project documents are tracked in the repository. Data files are automatically excluded using .gitignore files and server-side checks within GitLab. Programmers are responsible for ensuring that no sensitive data or log files are inadvertently included in their commits.

Once development is complete, programmers push their changes and initiate pull requests. All local changes are pushed to the remote GitLab repository, and frequent commits and pushes are encouraged. Upon completing a task, programmers create a pull request in GitLab to merge their changes into the main branch. These pull requests adhere to established guidelines, emphasizing small, focused changes with clear descriptions and thorough testing prior to submission. Merge conflicts are resolved before creating the pull request to ensure a smooth integration process. The lead programmer then reviews the pull request, addresses any outstanding issues, and approves the merge into the main branch, which is protected to prevent direct changes and maintain code integrity.

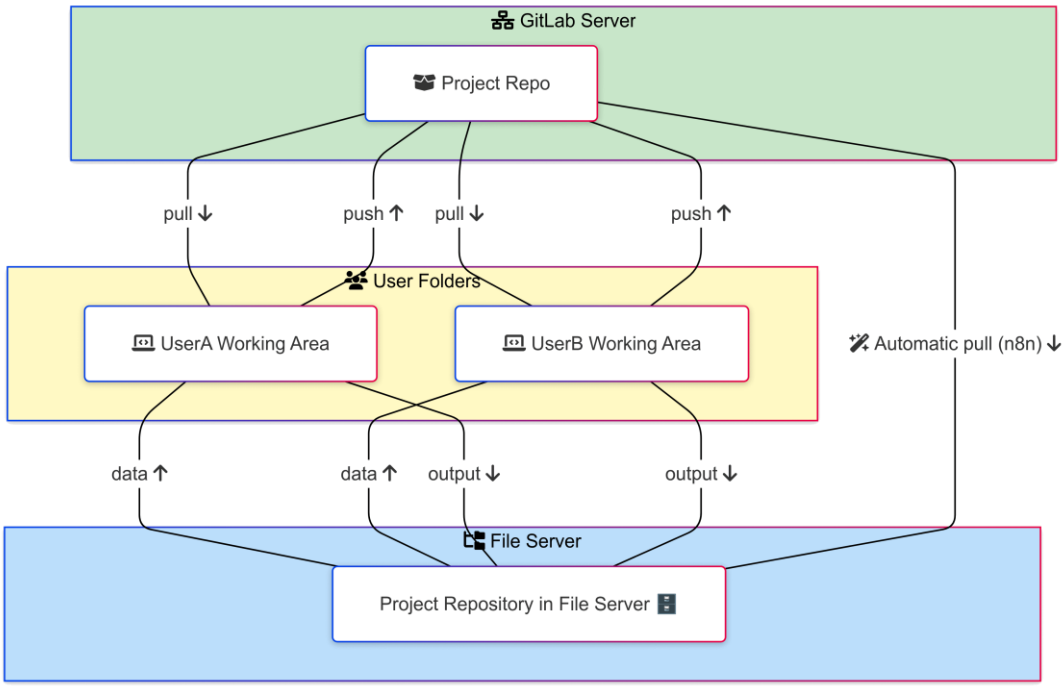

Figure 2: Interaction between GitLab server, user workspaces, and file server

QUALITY CONTROL CONSIDERATIONS

Integral to our Git workflow are stringent quality control measures. Output management is handled by the bootloader macro, ensuring a clear separation between development and production outputs. All development outputs are directed to a dedicated development area, effectively isolating them from production outputs and preventing accidental overwrites. This separation safeguards the integrity of production data and allows for iterative development without jeopardizing validated results. Furthermore, our validation process leverages Git's branching capabilities. Quality control (QC) programmers work in dedicated branches, enabling independent validation of code and output without interfering with ongoing development work. This parallel validation process ensures that code changes are thoroughly checked before being integrated into the main branch.

LESSONS LEARNED AND BEST PRACTICES

REFINING OUR APPROACH

Our pilot project highlighted several best practices for effective Git usage within our SAS environment. Employing efficient branching strategies, such as creating branches for users and merging regularly with the main branch proved crucial for organized and controlled development. Furthermore, adhering to clear branch naming conventions enhanced clarity and facilitated navigation through project history. Implementing robust commit management practices, including frequent, small commits with descriptive messages, and regularly pushing changes to the remote repository, significantly improved traceability and fostered seamless collaboration among team members.

KEY INFRASTRUCTURE ASPECTS

Several key technical considerations were essential for successful Git integration. Careful configuration of the self-hosted GitLab instance, including managing network permissions and establishing robust backup procedures, was critical for maintaining data integrity and security. Seamless integration with single sign-on (SSO), ensuring smooth file server connectivity, and adhering to established security protocols were all essential for a frictionless workflow. We also encountered limitations related to SAS and UNC paths. SAS Studio's built-in Git functionality proved insufficient for our needs, necessitating the use of a dedicated Git client to access the full range of Git features. Furthermore, compatibility issues with UNC paths prevented us from fully leveraging SAS's built-in Git capabilities.

FUTURE ENHANCEMENTS

We are committed to continually refining and expanding our Git implementation. Our ongoing efforts focus on increased automation of common tasks to further simplify the workflow for SAS programmers. We are actively exploring deeper integration with our existing toolset to streamline processes and enhance overall productivity. Automated updates to validation status on QC trackers, triggered by commit and pull request activity, are currently under development. We are also working to expand the adoption of Git to more complex projects within the organization. Finally, we are dedicated to continuously improving our training programs to empower SAS programmers with the Git expertise needed to thrive in this enhanced development environment.

CONCLUSION

Our journey of integrating Git into our SAS programming environment has been challenging yet ultimately transformative. By adopting a reasonable approach, conducting a thorough pilot, and prioritizing practical benefits over purely technical features, we have created a system that effectively serves the needs of our SAS programmers with the added advantage of version control using Git.

Several key lessons emerged from our experience. First, starting small and building confidence through a pilot project proved invaluable. This allowed us to identify potential challenges early on, refine our approach, and demonstrate the tangible benefits of Git to the wider team. Second, focusing on the practical advantages of Git, such as streamlined workflows and improved collaboration, was crucial for encouraging adoption. Third, actively listening to feedback from our SAS programmers and adapting our strategy accordingly ensured that the final implementation met their specific needs and addressed their concerns. Finally, while simplicity was a guiding principle throughout the implementation process, we also recognized the importance of innovation and custom macro exemplifies this approach.

Our successful Git implementation demonstrates that Git can be effectively tailored to the unique demands of SAS projects, paving the way for enhanced collaboration, improved code quality, and increased efficiency.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Rajwanur Rahman
Phastar
Bangladesh
Email: rajwanur.rahman@phastar.com

Brand and product names are trademarks of their respective companies.