US Professional Services and Delivery: Git Best Practices

Leveraging Git for Customer Engagements

SAS Institute Inc.

Version 1.0 26 January 2021

Contents

1 Overview 1
1.1 Workflow 1
1.2 Testing 1

2 Comparison to Subversion (SVN) 2
2.1 Common Commands 3
2.1.1 git clone 3
2.1.2 git branch 3
2.1.3 git checkout 3
2.1.4 git pull 4
2.1.5 git add 4
2.1.6 git commit 4
2.1.7 git push 4
2.2 Git Repository for USPS 5

3 Environment Setup 6
3.1 Request a Git Repository 6
3.2 Request Access to a Git Repository 6
3.3 Set Up Token Authentication 6
3.4 Set Up Playpen 8
3.5 Integrate with SAS Studio V 9
3.6 Playpen Directory Structure 12

4 Trunk-Based Methodology 13
4.1 Overview 13
4.2 Naming Conventions 13
4.2.1 Feature Branches 13
4.2.2 Bug Fix 13
4.2.3 Release Branch 13
4.3 Concepts 13
4.3.1 Merge Request 13
4.3.2 Feature Flagging (Toggling) 14
4.3.3 Cherry-Picking 14
4.4 Feature Flagging Workflow 14
4.4.1 Scenario 1: Adding a New Feature to the Existing Application 14
4.4.2 Scenario 2: Updating an Existing Feature 15
4.5 Merge Request Workflow 16
4.6 Submitting a Merge Request 16
4.7 Merge Request - Maintainer Perspective 18

5 Common Use Cases 21
5.1 New Code Development 21
5.2 Bug Fix 22
5.3 Promote to Dev Run User 24
5.4 New Feature Post-MVP 25
5.5 Bug Fix in Dev 27
5.6 Promotion to Test 28
5.7 Bug Fix in Test 30

5.8 Promotion to Production 31
5.9 Big Fix in Production 32
6 Additional Resources 34
Appendix A: Terms, Definitions, and Acronyms 35

Information about This Document

Document Control 36
Contacts 36
Revision History 36

Figures

Figure 1: GitLab User Icon 7
Figure 2: Adding SSH Key to GitLab 7
Figure 3: GitLab Your Projects Link 8
Figure 4: GitLab List of Projects 8
Figure 5: GitLab Clone Repository Link 9
Figure 6: SAS Studio V Git User Interface Icon 10
Figure 7: SAS Studio V Add Repository Link 10
Figure 8: SAS Studio V Configure Existing Repository 11
Figure 9: SAS Studio V Create Git Profile 11
Figure 10: Feature Forking with a New Feature 15
Figure 11: Feature Forking to Update Existing Feature 15
Figure 12: Merge Request Navigation Button 16
Figure 13: New Merge Request 16
Figure 14: Map Personal Repository to Main Repository 17
Figure 15: Dialog Form for Merge Request 17
Figure 16: Assignment Selection for Merge Request 17
Figure 17: Merge Request Submission Button 17
Figure 18: Merge Request Awaiting Action 18
Figure 19: Select the Desired Merge Request to Review 18
Figure 20: Merge Request Dialog Form 19
Figure 21: Merge Request Code Review Panel 19
Figure 22: Merge Request Overview Panel 20
Figure 23: Merge Request Acceptance Button 20
Figure 24: New Code Workflow 21
Figure 25: Quick Reference Legend 22
Figure 26: Bug Fix Workflow 23
Figure 27: Quick Reference Legend 23
Figure 28: Dev Promotion Workflow 24
Figure 29: Quick Reference Legend 25
Figure 30: New Feature Workflow 26
Figure 31: Quick Reference Legend 26
Figure 32: Bug Fix Post-Dev Deployment Workflow 27
Figure 33: Quick Reference Legend 28
Figure 34: Test Promotion Workflow 29
Figure 35: Quick Reference Legend 29

Figure 36: Bug Fix Post-Test Deployment Workflow 30
Figure 37: Quick Reference Legend 31
Figure 38: Prod Promotion Workflow 31
Figure 39: Quick Reference Legend 32
Figure 40: Bug Fix Post-Prod Promotion Workflow 33
Figure 41: Quick Reference Legend 33

Tables

Table 1: Git vs. SVN Commands 2
Table 2: USPS Git Repository Locations 5

1 Overview

The goal of this document is to provide developers with guidance on how to leverage Git within US Professional Services (USPS).

- Sections 2 and 3 describe how to request a Git repository for your project and how to integrate the repository into your programming environment.
- Section 4 details the trunk-based development paradigm followed by USPS when leveraging Git.
- Section 5 contains example workflows for various common tasks encountered by USPS in application development.
- Section 6 provides resources for additional information on using Git.

1.1 Workflow

Every project that leverages Git begins with a main Git repository for the project. This repository starts with a master branch that is the main code base for the project and serves as the repository that feeds the run account on the Development, Test, and Production servers. Maintainers of this repository peer review code before it can be merged into the master branch. The repository is cloned to a developer's playpen on the development server, where the developer is able to begin updating the code base. After the developer is ready to share their code, they push their code up to the repository where it can be retrieved by the rest of the development team. Under the trunk-based development paradigm, branches are created to add features or to fix bugs. When the branch is ready to be merged into the master branch, a merge request is created that prompts the maintainer(s) of the repository to peer review the changes before allowing the code to be merged.

1.2 Testing

The maintainer(s) of the repository peer review the changes requested by the pull request, prior to the code being merged into the master branch. Performance and quality assurance (QA) testing occur on the master branch, so that the overall system performance can be evaluated.

2 Comparison to Subversion (SVN)

One of the key differentiators between Git and Subversion (SVN) is the distributed nature of Git. With SVN, the repository resides on a centralized server that the developer engages by using SVN commands to push, pull, commit, and so on, the individual files that are then tracked on the SVN server.

Git, on the other hand, distributes the repository locally to a server each time the repository is cloned. For example, with the USPS implementations described in this document, each time a developer clones the repository to their individual playpen, Git not only retrieves the files within the project, but also creates hidden directories in the developer's playpen that allow version control to occur directly on the local server.

The benefit of the distributed Git approach is that version control can occur even when the developer is unable to access the remote Git server that resides within the hosted environment. The distributed nature of Git does require an additional step when compared to SVN. Whereas SVN code is sent to the remote server using the *svn commit* command, with Git, the *git commit* command commits the code to the local repository and a *git push* command is required to sync the local repository with the remote repository.

For further comparisons of SVN vs Git commands, refer to Table 1 (obtained from https://backlog.com/git-tutorial/reference/commands/).

Table 1: Git vs. SVN Commands

Command	Operation	Subversion
git clone	Copy a repository	svn checkout
git commit	Record changes to file history	svn commit
git show	View commit details	svn cat
git status	Confirm status	svn status
git diff	Check differences	svn diff
git log	Check log	svn log
git add	Addition	svn add
git mv	Move	svn mv
git rm	Delete	svn rm
git checkout	Cancel change	svn revert[1]
git reset	Cancel change	svn revert[1]
git branch	Make a branch	svn copy[2]
git checkout	Switch branch	svn switch
git merge	Merge	svn merge
git tag	Create a tag	svn copy[2]
git pull	Update	svn update
git fetch	Update	svn update
git push	It is reflected on the remote	svn commit[3]

[1]Revert in SVN is the cancel of change, but Revert in Git is the commit for negation. The meanings of Revert are different.

[2]Branch and tag are the same in the structure in SVN, but they are clearly different in Git

[3]SVN does not have the concept of local repository/remote repository, accordingly commit is directly reflected in the remote. However, Git has different reflecting methods for reflecting to the local repository and for reflecting to the remote repository.

2.1 Common Commands

2.1.1 git clone

2.1.1.1 Description

The *git clone* command clones a remote repository to a local file system, where the code can be developed and edited.

2.1.1.2 Syntax

git clone {url or ssh of remote repository} {local directory path}

2.1.2 git branch

2.1.2.1 Description

The *git branch* command can be used to identify branches, add a new branch, or delete an existing branch. When creating a new branch, that branch must then be checked out before it can be leveraged.

2.1.2.2 Syntax

Syntax to list the branch that is currently active in the local repository:

git branch

Syntax to list all the available branches:

git branch -a

Syntax to create a new branch:

git branch {new branch name}

Syntax to delete a branch:

git branch -d {branch name}

2.1.3 git checkout

2.1.3.1 Description

The *git checkout* command allows you to switch branches in your local repository.

2.1.3.2 Syntax

Syntax to checkout an existing branch:

git checkout {branch name}

Syntax to create a new branch and switch to it:

git checkout -b {new branch name}