Paper 114-2019

Git’r’done: Using Git in the Production SAS® Environment

Joe Matise, NORC at the University of Chicago

ABSTRACT

SAS® Professionals often need to use version control, such as Git, to manage their programs through the various stages of development and production release. In this talk, we will discuss the different options for using Git with SAS programs, including different ideas of the best way to use Git with different SAS clients and their respective file formats, and whether using the integrated Git functionality in different SAS clients is a good idea or not.

This should appeal to SAS developers at all levels who use version control in their daily work, and should not require any particular SAS knowledge.

INTRODUCTION

Git is a powerful solution for source code version control that is widely used in software development at all levels, from open-source software hosted on the popular Github repository to many Fortune 500 companies, and has an 88% marketshare among professional developers according to the 2018 Stack Overflow Developer Survey[1].

SAS Institute in recent years has added several hooks to its various software platforms to allow for easier Git integration, from adding built-in Git support in Enterprise Guide, SAS Studio, and Data Integration Studio to adding Git functions to the SAS data step language.

Beyond integration, SAS programs and projects can also be managed in Git the old fashioned way: through the command line or various first- or third-party Git GUI tools, such as SourceTree, GitKraken, or GitHub Desktop.

We will briefly introduce each of these methods of interacting with Git in SAS; going into some detail about using Git in Enterprise Guide (the most common use case, we feel) and more briefly in SAS Studio and Data Integration Studio, and then describe our typical workflow in SAS with regards to our production programs.

GIT IN ENTERPRISE GUIDE

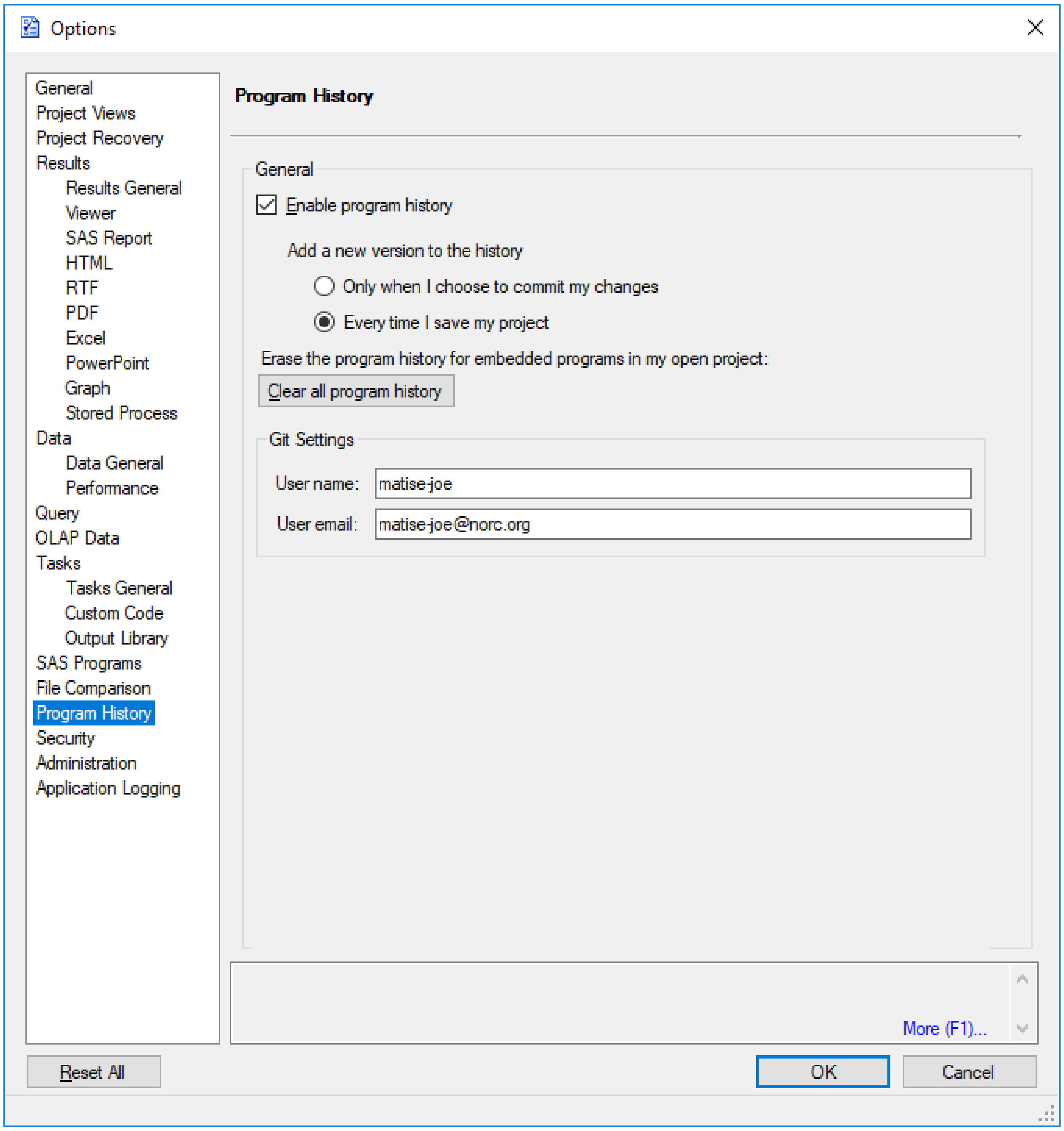
Enterprise Guide (EG) has included substantial Git integration starting with version 7.1:

- Programs embedded in the project are automatically tracked in an internal Git repository, and buttons in Enterprise Guide allow a user to commit changes, view the history of commits, and revert to earlier versions as appropriate.
- Programs stored outside of the project (in the file system) which are located in a Git repository can also use those integrated buttons to commit changes, view the history of commits, and revert to earlier versions.

INTERNAL REPOSITORY

Using the internal repository is as easy as using the save button. In Tools->Options, you will find a Program History tab:

[1] https://insights.stackoverflow.com/survey/2018#work-_-version-control , accessed 7/6/2019



Program History

General

[✓] Enable program history

Add a new version to the history

○ Only when I choose to commit my changes

[✓ Every time I save my project]

Erase the program history for embedded programs in my open project:

Git Settings

User name: matise-joe

User email: matise-joe@norc.org

On that tab, you can select to enable the program history, which enables the internal Git repository. You can choose to perform a commit every time you save, or only when you select “commit”. If you choose “Every time I save my project”, you get a long history, but you won’t get messages on your commits unless you use the Commit button – so if you choose this option, make sure to still commit occasionally using the button so you have more visibility of what the commits contain.

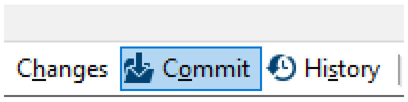
If you’re used to the idea of committing changes, using the commit-only option is probably best as it will yield a cleaner history with more comments.

EXTERNAL GIT REPOSITORY

Files that you link to, rather than embed, can be tracked with Git only if they are saved in a location that is tracked in an external Git repository. This may be confusing for users used to other systems (like SVN or CVS) which store the repository only on a server; Git, however, always stores the repository locally (and thus the change history). As such, SAS is able to directly modify that repository using the built-in Enterprise Guide functionality.

If your file is stored in an external repository, the commit functionality works largely the same as if you are using the internal repository. The only major difference is that you must include your user name and email address in the options when you enable program history tracking (while for internal storage it is optional), and you must use the commit button – commit on save is not enabled for external files.

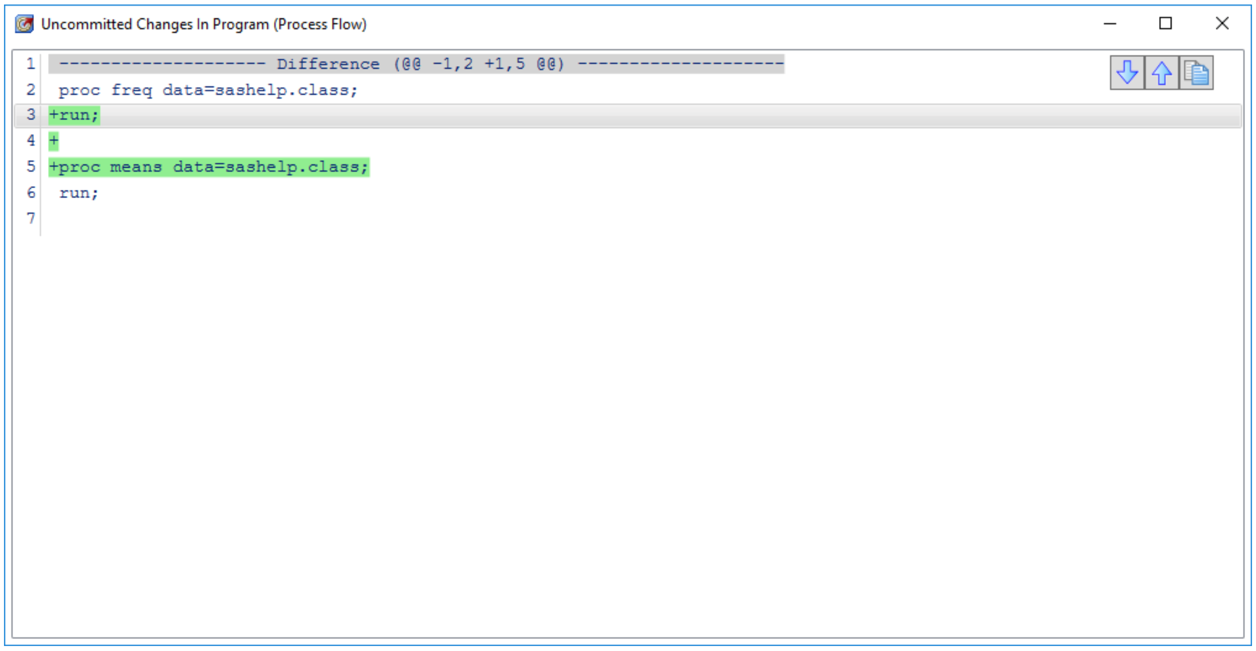
THE GIT TOOLBAR



The Git toolbar appears by default in the upper right portion of the Enterprise Guide toolbar when it is enabled. It contains three buttons: Changes, Commit (highlighted), and History.

Changes

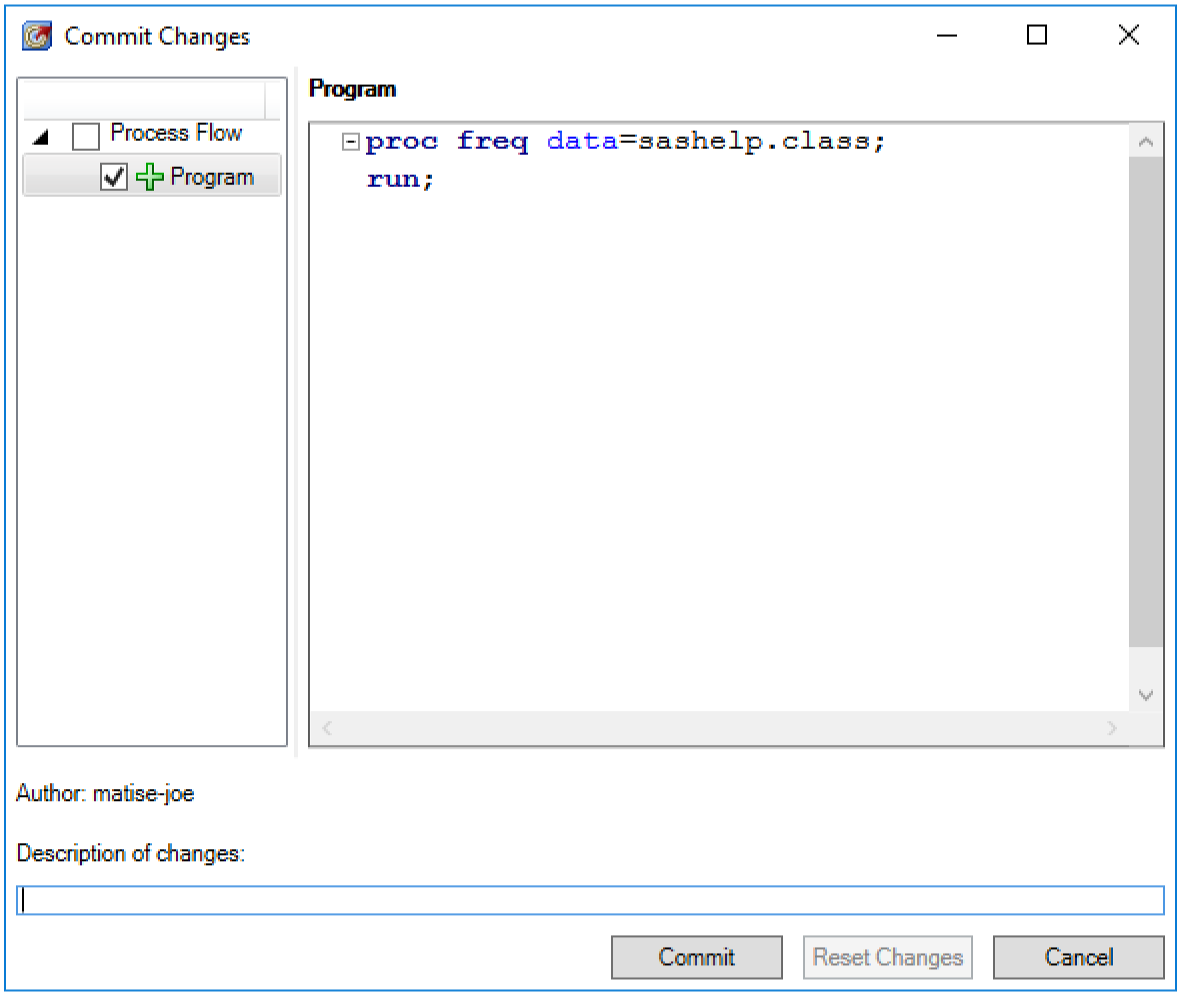
Changes shows you the difference between the program you are editing and what was last committed to the repository. Those changes are highlighted as below.



```
-------------------- Difference (@@ -1,2 +1,5 @@) --------------------
 proc freq data=sashelp.class;
+run;
+
+proc means data=sashelp.class;
 run;
```

Commit

Pressing Commit will show you a dialog box that allows you to see your changes, and add a description of those changes which will appear on the history entry for those changes. There is also a “Rest Changes” option on this screen, which will clear the changes made since the last commit.



```
proc freq data=sashelp.class;
run;
```

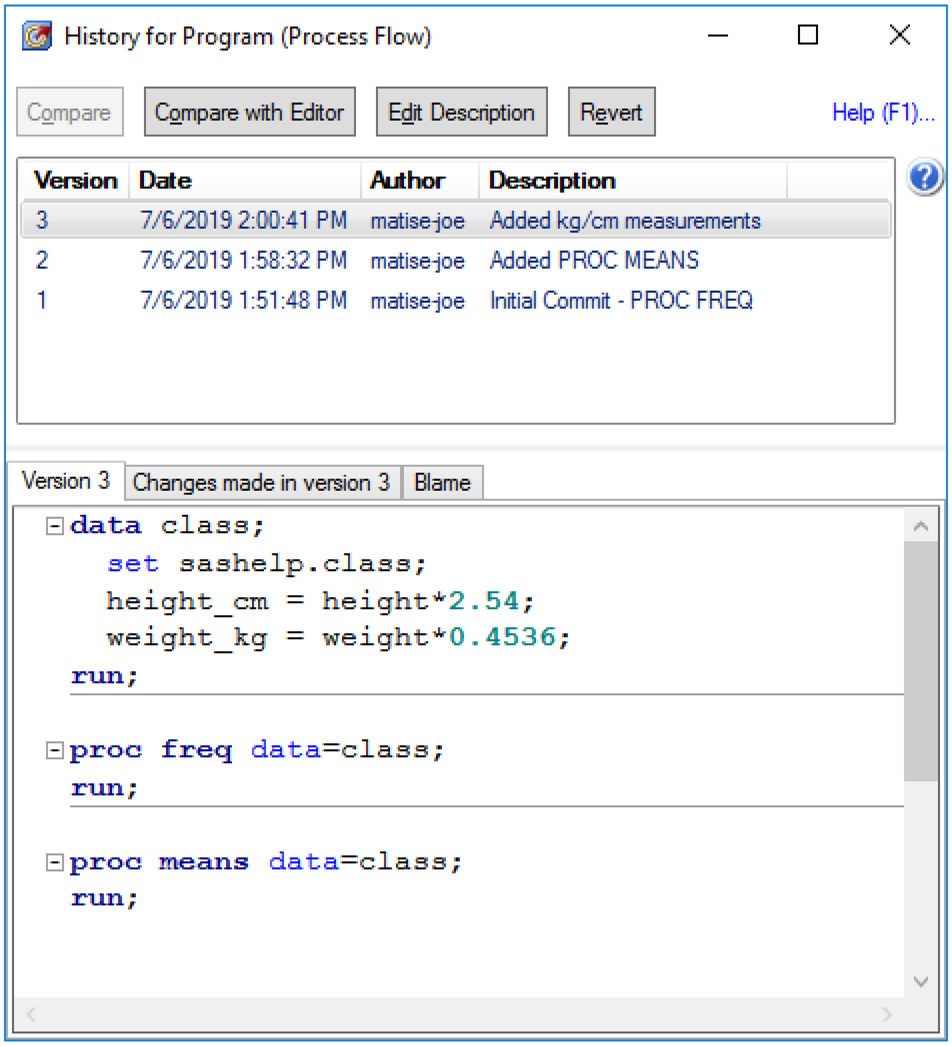
Author: matise-joe

Description of changes:

History

The History screen shows the list of all changes committed for the current program. Any description entered is shown, along with the user who committed the changes. The screen shows the current version, with a tab to show the changes made in that version, and a screen titled “Blame” which shows which commit each line came from.

This screen is also where the user can revert the file to an earlier version, as well as compare any two versions.



Version	Date	Author	Description
3	7/6/2019 2:00:41 PM	matise-joe	Added kg/cm measurements
2	7/6/2019 1:58:32 PM	matise-joe	Added PROC MEANS
1	7/6/2019 1:51:48 PM	matise-joe	Initial Commit - PROC FREQ

```
data class;
   set sashelp.class;
   height_cm = height*2.54;
   weight_kg = weight*0.4536;
run;

proc freq data=class;
run;

proc means data=class;
run;
```

Interacting with the server

Enterprise Guide does not include an integrated way to push to or pull from a Git server-based repository at the moment (such as Github or Bitbucket). In order to perform that step, an external client would be used.

WHAT'S MISSING FROM ENTERPRISE GUIDE

The biggest thing missing from Enterprise Guide is built-in interaction with the server Git repository (such as on Github, Bitbucket, etc.)– commits, pulls, etc. An external client is needed, and the same manipulation as with any other git project is needed outside of SAS. This is something fairly straightforward for a seasoned Git user, but likely take some time to get used to for a new Git user.

GIT IN SAS STUDIO

Git integration in SAS Studio was added in version 3.8, and is quite different from the integration with Enterprise Guide. While Enterprise Guide is limited to committing to the local repository, Studio is able to work with both the local repository and with the Git repository server, whether that be Github, Bitbucket, or similar.

SETTING UP GIT INTEGRATION IN SAS STUDIO

To use Git in SAS Studio, you need to perform a few setup tasks. These are listed here in outline, and are covered in greater detail in the SAS Studio 3.8 User's Guide[2].

- To authenticate, an SSH key file is required. Steps to create this are found in the User's Guide.
- Create a Git profile in SAS Studio. This can be done from the Preferences menu item. You will need to specify your SSH key location.
- Once a profile is created, set up a repository. You will need the URL of your repository, a local path to clone that repository to, and the profile you created above.

WORKING WITH GIT IN SAS STUDIO

Studio is able to do nearly all of the common Git functions directly in the interface, without needing to use another program or Git command line in most cases.

Cloning a repository

Cloning a repository is the first step to using Git. Select "Clone a Repository" under the Git section, and then enter the path to the git repository (for users of Bitbucket or similar Git repositories, this can be found in that interface by selecting to clone a repository from there). Assign a location for the Local folder to clone it to, and SAS will clone it to that location.

Committing changes

SAS supports commits directly in Studio. Under the Commit tab, files that are modified will appear in the Unstaged Files list. These files have different icons to show their status (such as a new file, a renamed file, a file with changes, a merge conflict, etc.). Committing is as easy as selecting these files and staging them (either by double-clicking or clicking the Stage button). Then, add a comment and press "Commit", and the changes will be committed.

Pulling and Pushing files from/to the server

Pulling and Pushing are directly supported with similarly-named buttons. Pulls that have merge conflicts will notify the user and move conflicted files to the Unstaged area.

Creating and Merging Branches

These both can be done from the History tab, by right-clicking on the commit to branch from or merge to. SAS Studio will recognize what branch you are on (and allow you to choose a different one as needed).

GIT IN DATA INTEGRATION STUDIO

Data Integration Studio (DIS) now supports Git in a limited fashion inside the tool. First, we will briefly mention the major setup requirements/steps to enable Git in DIS, and then we will discuss the functionality for the developer.

[2] *SAS® Studio 3.8: User's Guide*, "Understanding Git Integration in SAS Studio", available at https://documentation.sas.com/?docsetId=webeditorug&docsetTarget=p0hlfsbbbdnco4n1vep8z6k2p6us.htm&docsetVersion=3.8&locale=en , accessed 7/7/2019

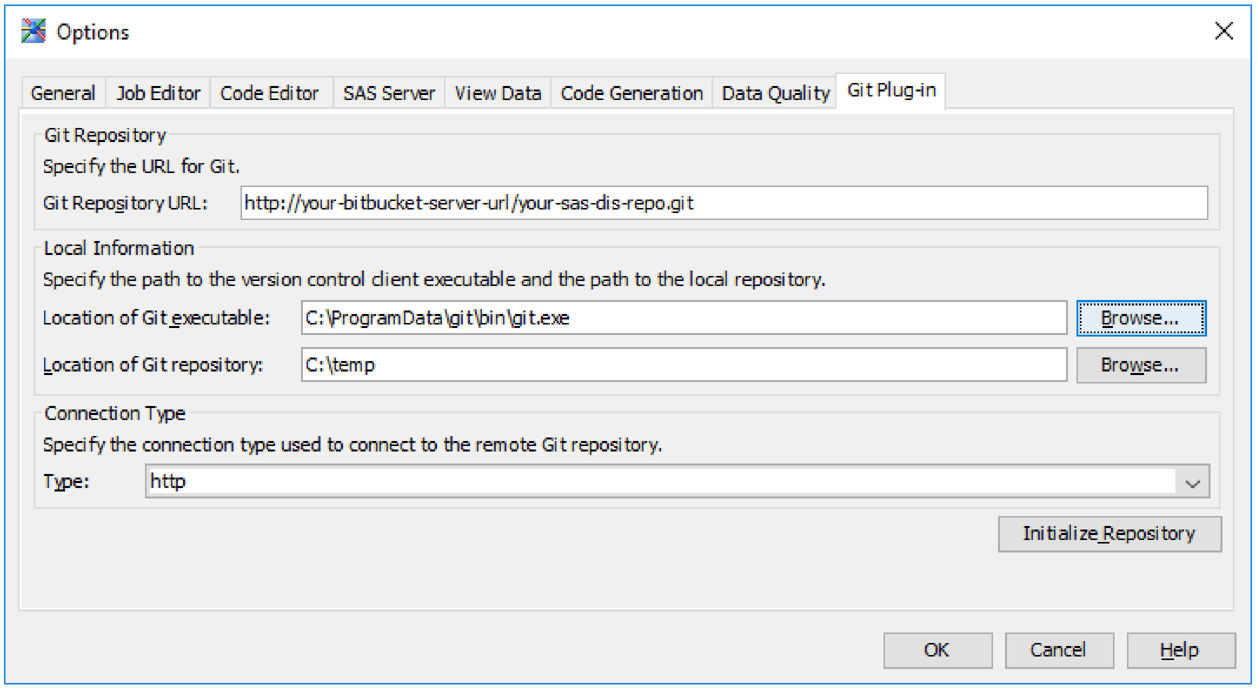
ENABLING GIT IN DATA INTEGRATION STUDIO

Git is available in DIS 4.904 and SAS 9.4 TS1M6, once a few setup steps have been completed. We will enumerate the key steps here, but for the complete steps please refer to the SAS documentation[3], or the excellent instructions posted in SAS Communities by Bogdan Teleuca[4].

- First, the CVS and SVN plugins must be removed from the SASHome folder, generally \SASHome\SASDataIntegrationStudio\4.7\plugins. If these are present, Git will not be shown as an option.
- You must have a Git repository accessible over the web that your server can connect to. This could be a Github repository, a Bitbucket repository, or potentially an on-site repository. You must also perform some setup tasks in that repository, including creating an Archives.xml file.
- You must have Git for windows installed on your local machine (The same machine SAS DIS is installed on)

Once you meet these requirements, you can proceed with the setup in DIS.

- In DIS, select Tools->Options, and you should see a tab for "Git Plug-in" if you correctly followed the above steps. Select that, and populate the Git Repository URL (from Bitbucket, Github, or similar), the location of git.exe, the location to store the local git repository (a temporary folder should work for this – SAS DIS will manage the clone and pull for you), and the connection method (HTTP or HTTPS – ask your administrator which to use).



Git Repository

Specify the URL for Git.

Git Repository URL: http://your-bitbucket-server-url/your-sas-dis-repo.git

Local Information

Specify the path to the version control client executable and the path to the local repository.

Location of Git executable: C:\ProgramData\git\bin\git.exe

Location of Git repository: C:\temp

Connection Type

Specify the connection type used to connect to the remote Git repository.

Type: http

[3] *SAS® Data Integration Studio 4.904: User's Guide*, "Example Setup for the GIT Repository", available at https://go.documentation.sas.com/?docsetId=etlug&docsetTarget=n044beoyw6bqben1gn0irr8nky8h.htm&docsetVersion=4.904&locale=fr , accessed 7/7/2019

[4] "SAS DI Developers: Unite! The new GIT plug-in in Data Integration Studio", available at https://communities.sas.com/t5/SAS-Communities-Library/SAS-DI-Developers-Unite-The-new-GIT-plug-in-in-Data-Integration/ta-p/549202, accessed 7/7/2019

USING GIT IN DATA INTEGRATION STUDIO

Git integration in Data Integration Studio is accomplished through the Archive Package utility. Jobs by default are not stored in Git until you Archive them. When you do so, if the Git plug-in is active, the job will be stored in your Git repository on the server directly, as a new commit. Note that this is more similar to the functionality of SVN/CVS than a typical Git usage – while there is a local copy of the repo, if you use this integrated functionality you will not interact with that local copy at any point, and all changes are made on the server from your point of view.

Creating a new job, or updating a job on the server

Right click on the job, select “Archive as a SAS Package”, give it a name, and enter a comment to help you identify the commit. You may have to enter your credentials (to the repository) at this step. When you click OK, if the archiving works properly you will see your job updated on the repository server as that name (.spk).

Comparing versions

If you wish to compare your current state to a version on the server, right click on the job and select “Archived SAS Packages’. This will give you a list of the past versions stored on the server. Find the version you want to compare, and select “Compare To….” This will show you the current version and the server version side by side, in XML form; while this is not the most user friendly way to browse the code, it does give you a good sense of exactly what changed between versions.

Restore a previous version

You will again use Archived SAS Packages here. If you still have the job in your folders, and are simply reverting to a previous version, then you can get here from the right-click menu on the job you are browsing; if not, and you need to restore a job that was deleted locally, you can get there from the File menu. Either way, you will want to use the “Import” option on that screen to bring your old version back onto the server.

Other supported functionality

There are a few other options, including deleting a SAS package, that are supported, and discussed in the earlier-mentioned documentation and instruction guide. Our recommendation would be to do most of your repository maintenance directly in the repository, though, as it will be simpler.

USING GIT IN DATA INTEGRATION STUDIO WITHOUT THE PLUG-IN

It is possible to use Git in a more traditional manner without the built-in plugin, though less practical in some ways. If you have a local Git repository created to store your code, you can Export your packages to that location, which will create a .spk file that contains your job. You can then commit this using your preferred Git tool, and manage it like you manage your other Git code.

However, you will not be able to (meaningfully) compare the versions, as that functionality is limited to the archived packages using the plug-in only.

You could also store the source code of your project in .sas files and version that, although depending on the complexity of your job it may not be possible to import that code back into a DIS job again.

CHALLENGES WITH THE GIT PLUG-IN WITH DATA INTEGRATION STUDIO

In our experience there are a few things that don’t work perfectly well right now, or that are not implemented quite as well as one would prefer.

- If you are in a multi-user environment, and checking jobs out to your personal folder (“My folder” or a similar location in DIS), the plug-in won’t properly recognize the package as the same as another user’s copy of that package. You can work around this, either by using a single folder for all commits (work on it in your personal folder, then copy to a common folder to do the commit, and import it from the same folder).

- The compare is very minimally useful. It's better than nothing, but it has a very steep learning curve to use, and for comparing user-written SAS code it is not very useful at all, as it puts the whole code on a single line (and then has very limited scrolling capability left-right). A better option might be to check "export steps as SAS code", and commit that separately, then do compares in a separate compare program.

All things considered, the plug-in is definitely a step up from having no Git options for Data Integration Studio, but we're hoping it gets some more attention from SAS to make the experience smoother and more useful.

GIT IN A PRODUCTION ENVIRONMENT: ONE EXAMPLE

At our company, some of our users use Git to manage code during development and in production deployments. Here is an example of how we use various components of SAS with Git.[5] We use Enterprise Guide as our primary development tool, Bitbucket for our Git server, Jira for issue tracking, and Stash for managing our repositories. The below is just one example workflow at our office; others make different choices, including using Git from the command line, or use Subversion (SVN) instead of Git.

DEVELOPMENT: SETTING UP A PROJECT FOR THE FIRST TIME

When we start the project, we will create a folder for it in the operating system that follows a consistent structure. This can be created by cloning a template project in Bitbucket into a new local folder, and then pointing that repository to a new project. This gives us the folder structure we expect to start off, and we can hit the ground running on the project.

During development, we create an Enterprise Guide project that will link to all of our programs and macros to simplify development. While we do not deploy the Enterprise Guide project itself into production, this is our preferred development platform and makes it easy to keep all of our programs accessible in one place.

Here we will generally create a process flow for each subfolder in the project, which can make it easier to find the programs in the operating system when we need to interact with them there.

DEVELOPMENT: COMMITTING/PUSHING/PULLING CHANGES

When we make changes, we can commit either inside Enterprise Guide or inside Stash. We often choose to use Stash for major commits, as we can commit and immediately push the changes to the server there, while we might use Enterprise Guide for smaller commits of a single file that we aren't ready to push to the server yet. We do not use Commit on Save due to a desire to maintain a cleaner commit history, but we regularly make manual commits, usually several times per day.

When we pull changes that others have made, we use Stash. This makes it easy to view the diffs and the change logs in one place.

DEVELOPMENT: FIXING BUGS, ADDING FEATURES

When we add a feature or fix a bug, we start by creating a branch in Jira. This allows us to track our changes just related to that feature or bug (or collection of bugs), and allows us to work collaboratively without stepping on each other's changes. Changes made in one branch do not affect other branches until they are merged, a process by which we each verify every change one by one.

Once we have created a branch, we use Stash to switch our local copy to the branch, and do a pull if needed to bring any new files down.

After we have made and tested our changes, we push the changes to Bitbucket. Then, a merge request is created from Jira, and our changes are reviewed by another developer. Once they have approved the merge request, we merge the changes into the master branch.

---

[5] This was written largely before SAS Studio-Git integration was widely available, and as such does not consider that as an option.

Finally, after the merge was completed, we use Stash to pull our changes down to the production code location.

DEVELOPMENT: DATA INTEGRATION STUDIO

Some of our workflows involve using Data Integration Studio to develop ETL jobs. We had to develop a new workflow when the Git plugin became available, as our prior workflow did not work well with Git due to the limitations of the plugin, but we feel like this method is satisfactory and not particularly difficult to adapt to.

When we work on a DIS package, we first check out the package by importing it from Git into our local user folder. This allows us to work on a package separately from other DIS developers who might need to access the same package, and avoids causing issues with production as well.

Once we are ready to commit, we archive that package back to the Git repository under our local user folder (which is a different location than we imported from in Git). This is partially due to the limitations of the plug-in, but it is also similar to normal Git workflow if you think of this as a branch. While we avoid having two developers work on the same package due to the complexities of merging code changes in DIS packages, this is still beneficial to segregate changes from production.

Once all of the changes to that package are completed, we are ready to merge. First, we view the changes in the local branch and another developer approves them. We then copy the package from our local folder to the Production folder (using normal DIS tools), and then archive that to the production Git repository. Finally, we view the changes and ensure there were no other changes in production since our branch was created.

PRODUCTION: DEPLOYING CODE TO PROD

Deploying new code to production with Git is very straightforward. Because we develop in Enterprise Guide but deploy .sas programs, we do have a few steps after development however to make the programs production ready.

First, we develop a wrapper program that %includes all of our modules, and performs any other necessary interface steps (such as parsing parameters that may be passed to the program from the command line). Then, we create a .bat file or .ps file (depending on our project's complexity) which will set environment variables and invoke SAS with the wrapper program. This .bat or .ps file (the launcher file) should be the location where custom parameters are defined (such as folder locations, round numbers, etc., which can vary without needing the production code to be modified). This .bat file is generally stored as a template, as it will be modified in production depending on the deployment requirements. We accomplish this by adding a .template extension to the program, and then anywhere it is deployed that program will be copied to a non-versioned program without the .template extension and then modified.

Once we have finished a module and have approved it for production, we use Stash to check out the Master branch to the production location. We then make the necessary changes to the launcher file and test it to make sure that the changes to that launcher file work in production. Finally, we add that launcher file to our scheduler.

As mentioned earlier, any further changes to production programs (bug fixes/feature requests) will be made in development and then merged to the master branch. After merging, Stash is used to pull the changes into the production location.

CONCLUSION

Git is a powerful development tool for not only tracking changes and enabling a user to view prior versions of code, but also for a collaborative work environment where multiple users work on the same code base. SAS provides several tools for simplifying Git integration, from Enterprise Guide to SAS Studio and Data Integration Studio. Using those tools in combination with third party tools and the Git command line, SAS developers can bring the power of Git to their workflow with relative ease.

ACKNOWLEDGMENTS

Thanks to Matt Kastin, who had the patience of Job while answering my many questions about Git, usually the same one three times. Thanks also to Jeff Vose and Dave Trevarthen for giving me much needed support and encouragement to write this paper and to present it.

RECOMMENDED READING

- *The SAS Dummy: Using built-in Git operations in SAS* (https://blogs.sas.com/content/sasdummy/2019/01/17/git-in-sas/)
- *SAS Functions to Drive Source Control with Git* (https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3057-2019.pdf)

CONTACT INFORMATION

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

Joe Matise
NORC at the University of Chicago
Email: matise.joe@gmail.com
https://stackoverflow.com/users/1623007/joe