Paper CT11

Using the source code control system GitHub to manage your SAS code

Frank Biedermann, Grünenthal GmbH, Aachen, Germany

ABSTRACT

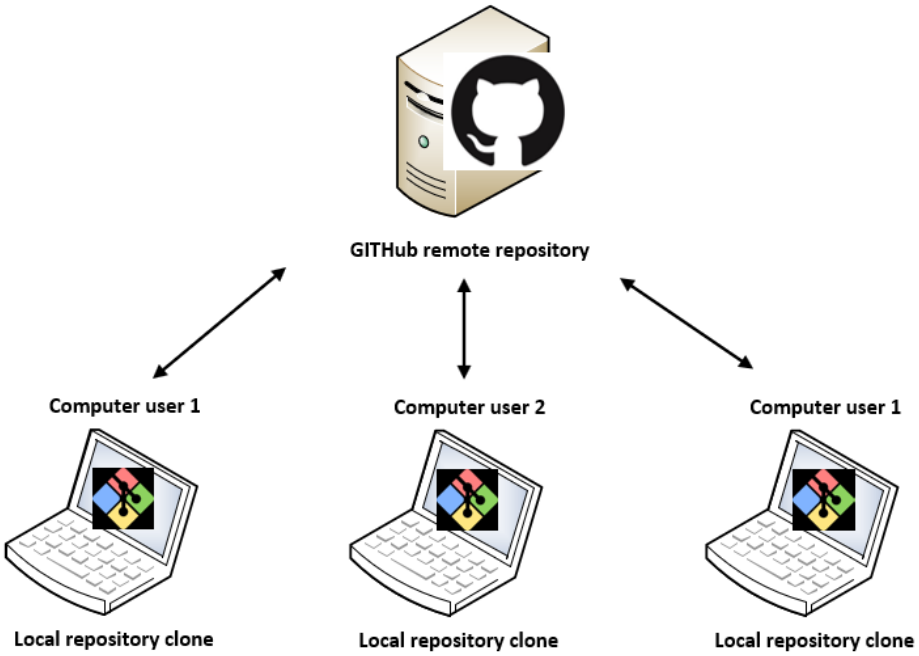
Since SAS® 9.4 Maintenance release 6, SAS added the Git infrastructure and functions. Additional Git client software is no longer needed. In this paper code snippets to CLONE, COMMIT, DIFF, PUSH and create a NEW_BRANCH in GitHub will be presented. Furthermore, examples will be shown how you can use GIT in the SAS Display Manager, SAS Studio and SAS Enterprise Guide.

WHAT IS GITHUB

This paper should not be understood as introduction to Git or GitHub itself. The paper can be more seen as a SAS GitHub tips and tricks sheet. But a little bit of Git / GitHub understanding is needed.

Let's start with Git. Git is an open source version control system, it is a version control system like SVN (Subversion) or CVS (concurrent versions system). Version control systems store programs, program revisions and modifications in a central repository. These features allow developers to download program code, make changes and upload the newest program code revision to a central repository. Git is a command line tool, but there are also programs available with a graphical user interface.

What is the main difference between GitHub and other version control systems? It is the hub - github.com where developers store their projects and network with like-minded people. When you create a new program code repository you can choose to make it public or private. Private repositories are accessible for you and the people you share them with. Public repositories are accessible for all github.com users. It is also possible to run a private Git server in your network.



TERMINOLOGY

Repository also called "repo": Each GitHub project has its own repository, all project-related files are stored here. Every repository has a unique URL (https, ftp or network).

Forking: Forking of a repository means creating a new project based on an existing GitHub project.

Commit: Commit save your program code changes to the local repository.

Pull or Merge request: You modified program code from an existing GitHub project and want to publish your changes. You can do this with a pull request, the author of the original repository can then see your changes. She/he can accept the changes (procedure is called merge), or can reject your changes (procedure is called close).

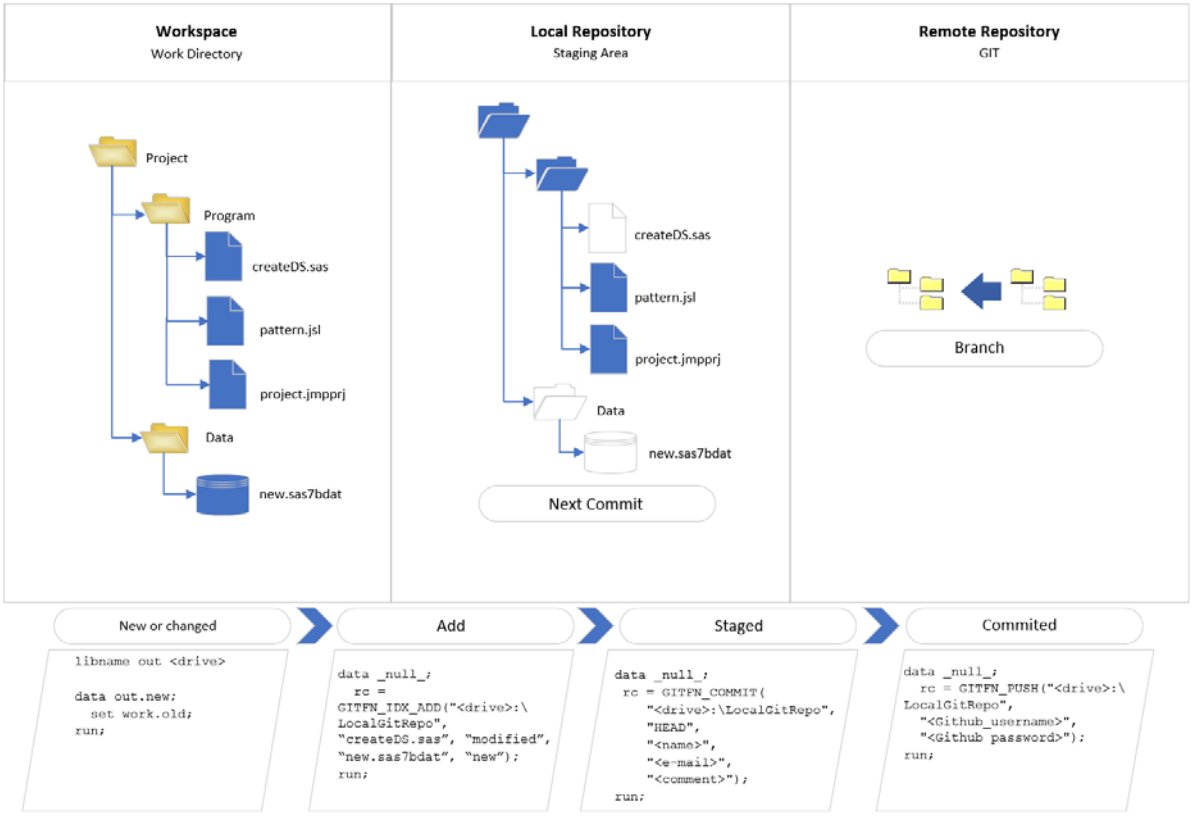
Changelogs: A changelog includes a chronological list of changes made by the developers. Who changed, what, when and where those files are stored.

BRANCHES

Branches are used to develop new features isolated from other branches. When creating a repository the master branch is the default branch. Other branches are used for developing, which will be merged to the master branch upon completion.

PROCESS

Git contains various areas with the possibility of tracking changes and keeping information about local and remote repositories.
A Git project contains three areas or states to keep track of the project work.
Workspace or working directory: the local project programming folder.
Local repository, staging or index area: where the next commit files will be stored.
Remote or Git repository: where the project metadata and new/modified files will be stored.



COPY AN EXISTING REPOSITORY TO YOUR COMPUTER

The function GITFN_CLONE clones a remote GitHub repository to a local computer or server folder.

```
data _null_;
      RC=GITFN_CLONE("https://github.com/Frank-Biedermann/phuse2019.git",
      "C:\tmp\git_phuse2019");
      put RC=;
run;
```

All examples in this paper are stored in the **public** GitHub repository phuse2019.git.
Because I use a public accessible GitHub repository the GITFN_CLONE function only requires two parameters: remote repository URL and target location. If you plan to use a private HTTPS or SSH repository which requires authentication then the following additional parameters are needed: user name, password, public SSH key path, private SSH key path.

PREPARE FILES FOR COMMITTING

The function GITFN_IDX_ADD prepares (stages) one or more of files for committing.

```
data _null_;
      RC=GITFN_IDX_ADD("C:\tmp\git_phuse2019",
            "createDS.sas", "new.sas7bdat", "new",
            "pattern.jsl", "project.jmpprj", "modified"
      );
run;
```

From the example: this function apparently has specific ‘nesting’ of parameter values if multiple files must be staged.

GITFN_IDX_ADD has three required parameters: - local repository path, - the filename(s) for staging, - status of the file(s): new or modified or deleted

COMMIT THE STAGED FILES TO THE LOCAL REPOSITORY

The function GITFN_COMMIT commits the previously prepared (staged) files.

```
data _null_;
  RC=GITFN_COMMIT("C:\tmp\git_phuse2019",
    "HEAD",
    "Frank Biedermann",
    "frank.biedermann@example.com",
    "first commit");
run;
```

GITFN_COMMIT has five required parameters: - local repository path, - update reference: HEAD commit to the head of the repository, - author name, - author e-mail address, - commit message

PUSH AND PULL

Use GITFN_PULL to pull online updates from other users and merges them with your local repository.

```
data _null_;
    RC=GITFN_PULL("C:\tmp\git_phuse2019");
run;
```

To connect to not SSH protected repositories only the parameter *local repository path* is required. SSH protected repositories require the following additional parameters: user name, password, path to the private SSH key, path to the public SSH key

The function uses the following return codes:

- Return Code 0: Successful pull with successful merge or fast-forward
- Return Code 1: Repository already up-to-date
- Return Code -1: An error occurred. Check the SAS log for more details

- Return Code -2: Conflicts occurred when merging the files pulled from the remote repository

GITFN_PUSH sends all committed files from a local repository to a remote Git repository.
The GITFN_PUSH function uses the currently configured remote repository.

```
data _null_;
	RC=GITFN_PUSH("C:\tmp\git_phuse2019",
		"Frank.Biedermann@example.com",
		"TOP SECRET");
run;
```

GITFN_PUSH function has the following required parameters:
a) For HTTPS connections: - local repository path, - user name, - password
b) Two more parameters are required for SSH connections: - path to the private SSH key, - path to the public SSH key

BRANCHES

CREATE A BRANCH

In simple words, a branch in Git is an individual project within a Git repository. Several branches can contain the same folders and files expect of some code line changes. Or these can contain completely different files and folders.

The function GITFN_NEW_BRANCH creates a new branch in the local repository.

```
data _null_;
	rc=GITFN_NEW_BRANCH("C:\tmp\git_phuse2019-1",
		"1d5b505f0381f90f664e6ffda9fd28e34e43fcff",
		"phuse2019-1",
		1);
run;
```

GITFN_NEW_BRANCH has four required parameters: - local repository path, - commit-id to create the branch on, - branch name, force - *overwrite existing branches with the same name or not to overwrite existing branches*.
Here an example if the branch already exists:
1 = true: `Branch "phuse2019-1" successfully created.`
0 =false: `ERROR: Return code from GIT is (4). failed to write reference 'refs/heads/phuse2019-1': a reference with that name already exists.`

The function does not have return codes. A successful branch creation returns a note to the SAS log.

CHECK OUT OR SWITCH A BRANCH

The function GITFN_CO_BRANCH checks a branch out to the local repository. You can use the function also to switch between branches.

```
data _null_;
	RC=GITFN_CO_BRANCH("C:\tmp\git_phuse2019-1", "phuse2019-1");
run;
```

GITFN_CO_BRANCH has two required parameters: local repository path, branch name

The following SAS note shows us that the repository successfully checked out:

```
NOTE: Branch "phuse2019-1" successfully checked out.
rc=0
```

MERGE A BRANCH

GITFN_MRG_BRANCH merges a GitHub branch with a checked-out branch. The merge result is stored in the local repository.

```
data _null_;
	rc=GITFN_MRG_BRANCH(
		"C:\tmp\git_phuse2019",
```

```
            "master",
            "Frank",
            "Frank.Biedermann@example.com");
run;
```

GITFN_MRG_BRANCH has four required parameters: - local repository path, - name of the branch you want to merge with the checked-out branch, - author name, - author e-mail

The function uses the following return codes:

- Return code: –2: Indicates that the index has conflicts. Check the log for conflicting files.
- Return code: –1: Indicates that the merge failed. Check the log for additional details.
- Return code: 0: Indicates that the merge was successful.
- Return code: 1: Indicates that the branch was already up-to-date.

DELETE A BRANCH

The function GITFN_DEL_BRANCH deletes a branch in the local repository.
0 = branch deleted, -1 any errors check the SAS log for additional information from Git.

```
data _null_;
   rc=GITFN_DEL_BRANCH(
      "C:\tmp\git_phuse2019",
      "phuse2019-1");
run;
```

GITFN_DEL_BRANCH has two required parameters: - local repository path, - branch name that you want to delete

STATUS OF REPOSITORY FILES

GET STATUS OBJECTS

The function GITFN_STATUS returns the number of status objects in your local repository and creates a status record. The function requires only the parameter: - local repository path

```
data _null_;
   N=GITFN_STATUS("C:\tmp\git_phuse2019");
run;
```

RETURN ATTRIBUTES FROM STATUS OBJECTS

The function GITFN_STATUS_GET returns attributes from status objects in your local repository. GITFN_STATUS must run before for to create the status objects.

```
data _null_;
      length _path $200.
             _status _staged $50.;
    rc=GITFN_STATUS_GET(1,"C:\tmp\git_phuse2019","Path",_path);
    rc=GITFN_STATUS_GET(1,"C:\tmp\git_phuse2019","Status",_status);
    rc=GITFN_STATUS_GET(1,"C:\tmp\git_phuse2019","Staged",_staged);
run;
```

GITFN_STATUS_GET has four required parameters: - an integer to loop over the structure of status objects, - local repository path, - requested attribute, - output variable

The function GITFN_STATUS_GET returns in this example only one status object, therefore the integer value 1.

GITFN_STATUS_GET returns the following SAS log message:

```
NOTE: Entry: gitfn_status.sas. Staged: False. Status: New.
```

CLEAR STATUS OBJECTS

The GITFN_STATUSFREE clears the status record object that was returned by the function GITFN_STATUS. Only the parameter: - local repository path is required.

```
data _null_;
    rc=GITFN_STATUSFREE("C:\tmp\git_phuse2019");
run;
```

Function return codes are:

- Return code: 0 status record was successfully cleared.
- Return code: 1 no status record was found for the specified repository; no status record could be cleared.

GET NUMBER OF COMMITED OBJECTS

GET COMMIT LOG

The function GITFN_COMMIT_LOG returns the number of committed objects in the local repository.
The function requires only the parameter: - local repository path

```
data _null_;
       n = GITFN_COMMIT_LOG("C:\tmp\git_phuse2019");
run;
```

The put n= prints the number of committed objects in the repository in the SAS log.

GET COMMIT OBJECTS

GITFN_COMMIT_GET returns a specified attribute from a commit object in the local repository.
GITFN_COMMIT_LOG must run before to get the commit objects.

```
data _null_;
    length commit_id author_name $200;
    rc=GITFN_COMMIT_GET(1,"C:\tmp\git_phuse2019", "id", commit_id);
    rc=GITFN_COMMIT_GET(1,"C:\tmp\git_phuse2019", "author", author_name);
    put commit_id=;
    put author_name=;
run;
```

GITFN_COMMIT_GET has four required parameters: - an integer to loop over the structure of commit objects, - local repository path, - requested attribute, - output variable

The function GITFN_COMMIT_GET returns in this example only one committed object, so the integer value is 1.

GITFN_COMMIT_GET returns the following SAS log message:

```
commit_id=0d15a34b3a2728480d88166295530d3c2c6b86d2
author_name=Frank Biedermann
```

CLEAR COMMIT OBJECTS

The GITFN_ COMMITFREE clears the commit record object that was returned by the function GITFN_COMMIT_GET. Only the parameter: - local repository path is required.

```
data _null_;
    rc=GITFN_COMMITFREE("C:\tmp\git_phuse2019");
run;
```

Function return codes are:
- 0 commit record was successfully cleared.
- 1 no commit record was found for the specified repository; no commit record could be cleared.

COMPARE TWO COMMITS

RETURN THE NUMBER OF CHANGES

The function GITFN_DIFF returns the number of changes between two commits and creates a diff record object in the local repository.

```
data _null_;
    n=gitfn_diff(
       "C:\tmp\git_phuse2019",
       "0d15a34b3a2728480d88166295530d3c2c6b86d2",
       "cf70de661bf032d4c1d0a2c8d32bbe2f366cd15d");
run;
```

GITFN_DIFF has three required parameters: - local repository path, - older commit id, - newer commit id

The put n= returns the number of changes between the two commits in the SAS log.

RETURN ATTRIBUTES FROM A DIFF OBJECT
The function GITFN_DIFF_GET returns the specified attribute (file, diff_content, diff_type) from a diff object in a local Git repository. GITFN_DIFF must run before for to get the commit objects.

```
data _null_;
   length _file_path $ 200;
   rc=gitfn_diff_get(1,
      "C:\tmp\git_phuse2019",
      "0d15a34b3a2728480d88166295530d3c2c6b86d2",
      "cf70de661bf032d4c1d0a2c8d32bbe2f366cd15d",
      "file",
      _file_path);
   put _file_path=;
run;
```

GITFN_DIFF_GET has six required parameters: - an integer to loop over the structure of commit objects, - local repository path, - older commit id, - newer commit id, - requested attribute, - output variable

The diff content is returned in JSON format.

CLEAR DIFF RECORD
The GITFN_ DIFF_FREE function clears the diff record that was returned by the function GITFN_DIFF.

```
data _null_;
    rc=gitfn_diff_free(
       "C:\tmp\git_phuse2019",
       "0d15a34b3a2728480d88166295530d3c2c6b86d2",
       "cf70de661bf032d4c1d0a2c8d32bbe2f366cd15d");
run;
```

GITFN_ DIFF_FREE has three required parameters: - local repository path, - older commit id, - newer commit id

Function return codes are:
- Return code: 0 diff record was successfully cleared.
- Return code: 1 no diff record was found for the specified repository; no diff record could be cleared.

MODIFIED FILE COMPARE
The function GITFN_DIFF_IDX_F returns changes made to a file in the index against the last committed repository version of that file. Changes will be stored in a specified variable. The output of the diff functions is limited to 32,767 characters. It can spark off that the output is not completed displayed.

The function has three required parameters: - local repository path, - file name from the file that you want to diff, - variable where the content of the diff will be stored

```
data _null_;
    length _diff_content $ 32767;
    _diff_content="";
    rc=GITFN_DIFF_IDX_F("C:\tmp\git_phuse2019",
```

```
        "gitfn_idx_add.sas",
        _diff_content
    );
    put rc=
    put _diff_content=;
run;
```

GITFN_DIFF_IDX_F returns the following SAS log message:

```
NOTE: _diff_content ={ diff-content }
```

And has the following return codes:

- Return code: 0 Indicates the function was successful.
- Return code: -1 Indicates the function failed. Additional log message: `ERROR: Unable to diff file "<file name>" because it's not currently in the index or it's a new file`

CONCLUSION

GIT provides fast and comfortable ways to track programming projects, whether the project is by one programmer, or a team of programmers. GIT is easily accessible for single programmers and small projects that need some kind of tracking capabilities, in addition GIT has many complex features available. With SAS 9.4 Maintenance release 6 these features are available in the SAS program editor and can be use without additional GIT client software.

REFERENCES

GitHub Guides: https://guides.github.com/
SAS Documentation: https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=n1mlc3f9w9zh9fn13qswiq6hrta0.htm&docsetVersion=9.4&locale=en

RECOMMENDED READING

SAS Functions to drive source control with GIT by Danny Zimmerman
https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3057-2019.pdf
An Introduction to GIT Version Control for SAS Programmers by Stephen Philp
https://www.lexjansen.com/wuss/2012/94.pdf

CONTACT INFORMATION

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

Frank Biedermann
Grünenthal GmbH
Aachen / 52099
Email: Frank.Biedermann@grunenthal.com
Web: www.grunenthal.com