Verwendung des Versionskontrollsystems Git für SAS Programmcode

Frank Biedermann
Grünenthal GmbH
Zieglerstraße 6
52078 Aachen
Frank.Biedermann@grunenthal.com

Zusammenfassung

Seit SAS 9.4 Maintenance Release 6 ist es möglich, das Versionskontrollsystem Git ohne einen zusätzlich installierten Git Client direkt vom SAS Code aus anzusteuern. In diesem Vortrag werden zum Beispiel Programmcodes für die folgenden Git Befehle gezeigt: GET, CLONE, COMMIT, DIFF, PUSH sowie das Erstellen einer neuen Branch in einem Git Code Repository. Des Weiteren werden Beispiele gezeigt, wie Git im SAS Display Manger, SAS Studio und im SAS Enterprise Guide verwendet werden kann.

Schlüsselwörter: GIT, GIT_PULL, GIT_CLONE, GIT_KOMMIT, GIT_DIFF, GIT_PUSH, GIT_STATUS, BRANCHING

1 Was ist GitHub?

Dieses Dokument soll nicht als Einführung in Git oder GitHub selbst verstanden werden. Das Dokument kann eher als ein SAS GitHub-Tipps und -Tricks Beitrag gesehen werden. Jedoch ist ein wenig Git/GitHub-Verständnis erforderlich.

Beginnen wir mit Git. Dies ist ein Open-Source-Versionskontrollsystem, es ist ein Versionskontrollsystem wie SVN (Subversion) oder CVS (Concurrent Versions System). Versionskontrollsysteme speichern Programme, Programmrevisionen und Modifikationen in einem zentralen Repository (SVN, CVS) oder in einem dezentralen Repository (Git). Diese Funktionen ermöglichen es Entwicklern, Programmcodes herunterzuladen, Änderungen vorzunehmen und die neueste Programmcoderevision in ein Repository hochzuladen. Git ist ein Kommandozeilenwerkzeug, aber es gibt auch Programme mit einer grafischen Benutzeroberfläche.

Was ist der Hauptunterschied zwischen GitHub und anderen Versionskontrollsystemen? Es ist der Hub - github.com, dort wo Entwickler ihre Projekte speichern und sich mit Gleichgesinnten vernetzen. Wenn Entwickler ein neues Programmcode-Repository erstellen, dann können sie wählen, ob es öffentlich oder privat sein soll. Private Repositories sind nur für die Ersteller des Repositories und derer Personen zugänglich, mit denen sie es teilen. Öffentliche Repositories sind für alle Benutzer von github.com zugänglich. Es ist auch möglich, einen privaten Git-Server in einem privaten oder Unternehmensnetzwerk zu betreiben.

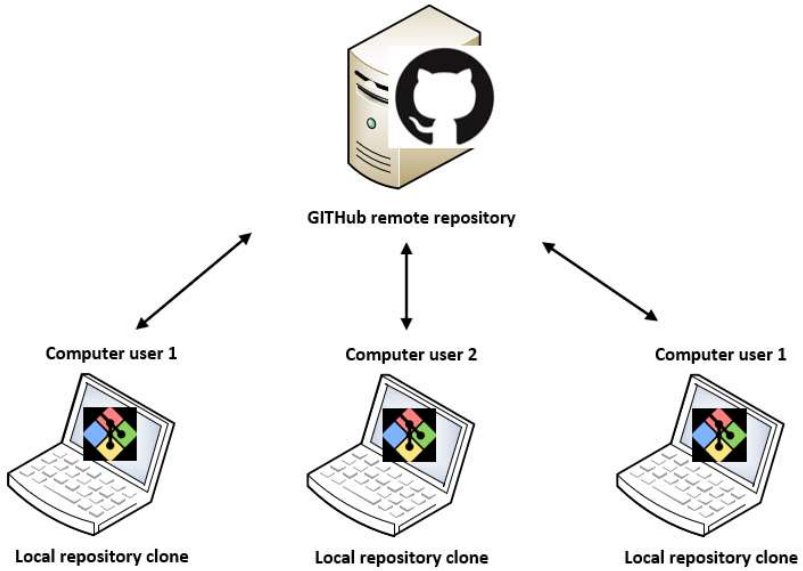

Abbildung 1: Git-Konzept

2 Terminologie

Repositories auch "Repos" genannt: jedes GitHub-Projekt hat sein eigenes Repository, alle projektbezogenen Dateien werden hier gespeichert. Jedes Repository hat eine eindeutige URL (https, ftp oder Netzwerk).

Forking: forking eines Repositories bedeutet, dass ein neues Projekt auf der Grundlage eines bestehenden GitHub-Projekts erstellt wird.

Commit: commit speichert Programmcode Änderungen im lokalen Repository.

Pull oder Merge Anfrage: Sie haben Programmcode aus einem bestehenden GitHub-Projekt geändert und möchten Ihre Änderungen veröffentlichen. Sie können dies mit einer Pull-Anforderung tun, der Autor des ursprünglichen Repositories kann dann Ihre Änderungen sehen. Sie/er kann die Änderungen akzeptieren (das Verfahren wird als Merge bezeichnet), oder Ihre Änderungen ablehnen (das Verfahren wird als Close bezeichnet).

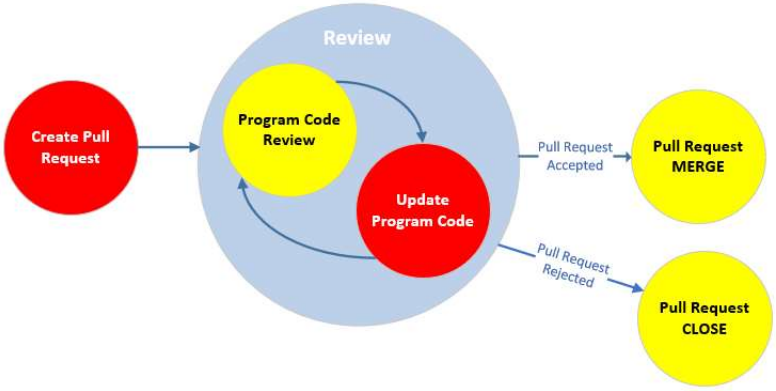

Abbildung 2: Pull/Merge Anfrage

Changelogs: Ein Changelog enthält eine chronologische Liste der von den Entwicklern vorgenommenen Änderungen. Wer, was, wann und wo diese Dateien geändert wurden.

3 Branches

Branches werden verwendet, um neue Funktionen isoliert von anderen Branches zu entwickeln. Beim Anlegen eines Repositories ist die Master Branch die Default Branch. Andere Branches werden für die Entwicklung verwendet, welche nach Fertigstellung in die Master Branch überführt werden.

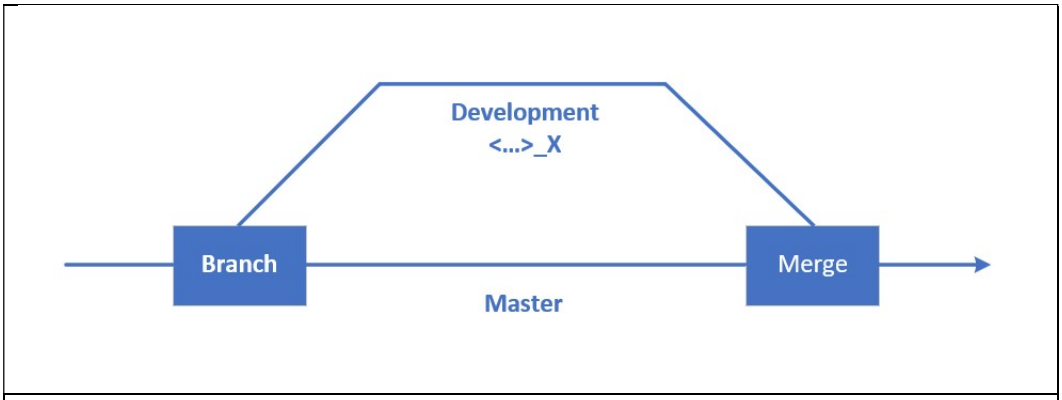

Abbildung 3: Branches

4 Prozess

Git enthält verschiedene Möglichkeiten, Änderungen zu verfolgen und Informationen über lokale und entfernte Repositories zu speichern.
Ein Git-Projekt enthält drei Bereiche oder Zustände, um die Projektarbeit zu verfolgen.
Workspace or working directory: ist der lokale Projekt-Programmierordner.
Local repository, staging oder index area: ist der Ort, an dem die nächsten Commit-Dateien gespeichert werden.
Remote- oder Git-Repository: ist der Ort, an dem die Metadaten des Projekts und neue/geänderte Dateien gespeichert werden.

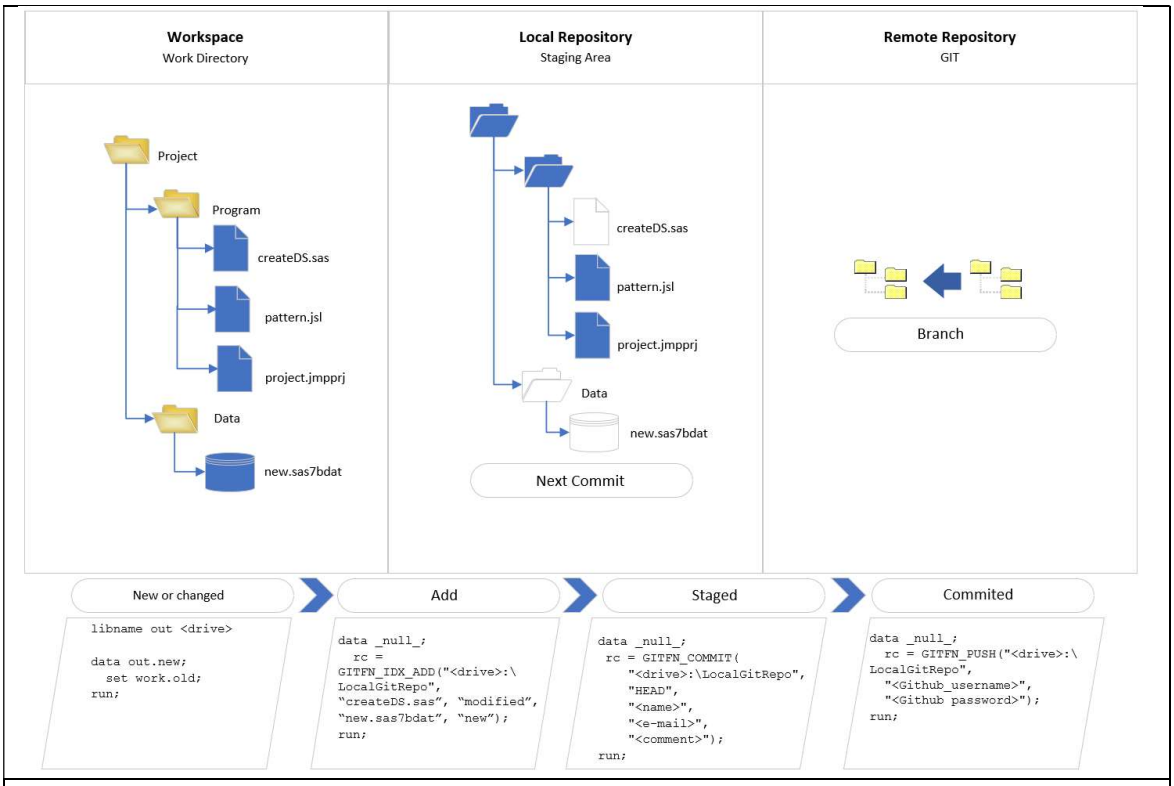

Abbildung 4: Prozess

4.1 Kopieren eines vorhandenen Repositories auf den Computer

Die Funktion GITFN_CLONE klont ein entferntes GitHub-Repository und speichert dieses auf einen lokalen Computer oder in einem Server-Ordner.

```
data _null_;
     RC=GITFN_CLONE(
          "https://github.com/Demo-Benutzer/ksfe2020.git",
          "C:\tmp\git_ksfe2020");
     put=RC;
run;
```

In diesem Beispiel wird ein öffentlich zugängliches GitHub-Repository verwendet. Die Funktion GITFN_CLONE benötigt nur zwei Parameter: die URL des Remote-Repositories und den Zielort. Wenn Sie planen, ein privates HTTPS- oder SSH-Repository zu verwenden, das eine Authentifizierung erfordert, sind folgende zusätzliche Parameter erforderlich: Benutzername, Kennwort, öffentlicher SSH-Schlüsselpfad, privater SSH-Schlüsselpfad.

4.2 Dateien vorbereiten für einen Commit

Die Funktion GITFN_IDX_ADD bereitet (stages) 1 bis N Anzahl von Dateien für den COMMIT Prozess vor.

```
data _null_;
     RC=GITFN_IDX_ADD("C:\tmp\git_ksfe2020",
          "createDS.sas", "new.sas7bdat", "new",
          "pattern.jsl", "project.jmpprj", "modified"
     );
     put RC=;
run;
```

GITFN_IDX_ADD hat drei erforderliche Parameter:

- lokaler Repository-Pfad
- den/die Dateinamen für die Bereitstellung
- Status der Datei(en): neu, geändert, gelöscht

4.3 Dateien die im Stage-Bereich sind an das lokale Repository übergeben

Die Funktion GITFN_COMMIT überträgt die von der Funktion GITFN_IDX_ADD vorbereiteten Dateien (staged).

```
data _null_;
  RC=GITFN_COMMIT(
    "C:\tmp\git_ksfe2020",
    "HEAD",
    "Dummy Name",
    "dummy.name@example.com",
    "first commit");
   put RC=;
run;
```

GITFN_COMMIT hat fünf erforderliche Parameter:

- lokaler Repository-Pfad
- Update-Referenz: HEAD Commit an den Leiter des Repositoriums
- Autorenname
- E-Mail-Adresse des Autors
- Commit-Nachricht

4.4 Push und Pull

GITFN_PULL kann verwendet werden, um Online-Updates von anderen Benutzern abzurufen und sie mit Ihrem lokalen Repository zusammenzuführen.

```
Data _null_;
    RC=GITFN_PULL("C:\tmp\git_ksfe2020");
    put RC=;
run;
```

Für die Verbindung zu nicht SSH-geschützten Repositories ist nur der Parameter „Lokaler Repository Pfad“ erforderlich. SSH-geschützte Repositories erfordern die folgenden zusätzlichen Parameter:

- Benutzername
- Kennwort
- Pfad zum privaten SSH-Schlüssel
- Pfad zum öffentlichen SSH-Schlüssel

Die Funktion gibt die folgenden Codes zurück (SAS Log):

- 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 sendet alle übergebenen Dateien von einem lokalen Repository an ein remote Git-Repository.
Die Funktion GITFN_PUSH verwendet das konfigurierte Remote-Repository als aktuelle Branch.

```
data _null_;
     RC=GITFN_PUSH(
          "C:\tmp\git_ksfe2020",
          "dummy.name@example.com",
          "TOP SECRET"
     );
     put RC=;
run;
```

Die Funktion GITFN_PUSH hat die folgenden erforderlichen Parameter:
Für HTTPS-Verbindungen:

- lokaler Repository-Pfad
- Benutzername
- Kennwort

Für SSH-Verbindungen sind zwei weitere Parameter erforderlich:

- Pfad zum privaten SSH-Schlüssel
- Pfad zum öffentlichen SSH-Schlüssel
-

5 Branching

5.1 Erstellen einer Branch

Mit einfachen Worten: Eine Branch in Git ist ein einzelnes Projekt innerhalb eines Git-Repositories. Mehrere Branches können die gleichen Ordner und Dateien enthalten, ausgenommen von ein paar Codezeilenänderungen. Oder sie können völlig unterschiedliche Dateien und Ordner beinhalten.
In SAS wird mit der Funktion GITFN_NEW_BRANCH ein neue Branch im lokalen Repository angelegt.

```
data _null_;
     rc=GITFN_NEW_BRANCH(
          "C:\tmp\git_ksfe2020-1",
          "1d5b505f0381f90f664e6ffda9fd28e34e43fcff",
          "ksfe2020-1",
          1
     );
     put rc=;
run;
```

GITFN_NEW_BRANCH hat vier erforderliche Parameter:
- lokaler Repository-Pfad
- Commit-id zum Erstellen der Branch
- BRANCH Name
- force - überschreibt eine bestehende Branch mit demselben Namen

Hier ein Beispiel, wenn die Branch bereits existiert:
1 = true: `Branch "ksfe2020-1" successfully created.`
0 = false: `ERROR: Return code from GIT is (4). failed to write reference 'refs/heads/ksfe2020-1': a reference with that name already exists.`
Die Funktion hat zwei mögliche Rückgabecodes: 0 = erfolgreich, -1 = jeder Fehler der auftritt.

5.2 Auschecken (Wechsel) einer Branch

Mit der Funktion GITFN_CO_BRANCH können Git Branches in das lokale Repository ausgecheckt werden. Die Funktion kann auch dafür verwendet werden, um zwischen Branches zu wechseln.

```
data _null_;
     RC=GITFN_CO_BRANCH(
          "C:\tmp\git_ksfe2020-1", "ksfe2020-1"
     );
     put=RC;
run;
```

GITFN_CO_BRANCH hat zwei erforderliche Parameter:
- lokaler Repository-Pfad
- BRANCH Name

Die folgende SAS Note zeigt uns, dass das Repository erfolgreich ausgecheckt wurde:

```
NOTE: Branch "ksfe2020-1" successfully checked out.
rc=0
```

5.3 Zusammenführen von Branches

GITFN_MRG_BRANCH führt (MERGE) eine GitHub Branch mit einer ausgecheckten Branch zusammen.

```
data _null_;
     rc=GITFN_MRG_BRANCH(
          "C:\tmp\git_ksfe2020",
          "master",
          "Dummy name",
          "dummy.name@example.com "
     );
     put rc=;
run;
```

GITFN_MRG_BRANCH hat vier erforderliche Parameter:
- lokaler Repository-Pfad
- Name der Branch, welche mit dem ausgecheckten Branch zusammengeführt werden soll
- Autorenname
- Autor-E-Mail

Die Funktion verwendet die folgenden Rückgabecodes:
- 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.

5.4 Eine Branch löschen

Die Funktion GITFN_DEL_BRANCH löscht eine Branch im lokalen Repository.
0 = BRANCH gelöscht, -1 Fehler prüfe die SAS Log auf zusätzliche Informationen aus Git.

```
data _null_;
   rc=GITFN_DEL_BRANCH(
      "C:\tmp\git_ksfe2020",
      "ksfe2020-1"
   );
    put rc=;
run;
```

GITFN_DEL_BRANCH hat zwei erforderliche Parameter:

- lokaler Repository-Pfad
- Name der Branch die gelöscht werden soll

6 Status der Repository-Dateien

6.1 Ausgabe von Status Objekten

Die Funktion GITFN_STATUS gibt die Anzahl der Statusobjekte in einem lokalen Repository zurück und erstellt eine Variable, die einen Statussatz enthält. Die Funktion benötigt nur den Parameter: lokaler Repository-Pfad

```
data _null_;
   N=GITFN_STATUS(
     "C:\tmp\git_ksfe2020"
   );
   put n=;
run;
```

6.2 Rückgabe-Attribute von Status Objekten

Die Funktion GITFN_STATUS_GET gibt Attribute von Statusobjekten in einem lokalen Repository zurück. GITFN_STATUS muss zuvor ausgeführt werden, um einen Statussatz anzulegen.

```
data _null_;
     length _path $200.
            _status _staged $50.;
    rc=GITFN_STATUS_GET(1,"C:\tmp\git_ksfe2020","Path",_path);
    rc=GITFN_STATUS_GET(1,"C:\tmp\git_ksfe2020","Status",_status);
    rc=GITFN_STATUS_GET(1,"C:\tmp\git_ksfe2020","Staged",_staged);
run;
```

GITFN_STATUS_GET hat vier erforderliche Parameter:

- Anzahl Loops
- lokaler Repository-Pfad
- Objekt-Attribut
- Ausgabevariable

Die Funktion GITFN_STATUS_GET gibt in diesem Beispiel nur ein Statusobjekt zurück, daher ist der Wert 1 für die variable Anzahl der Loops.
GITFN_STATUS_GET gibt die folgende SAS-Protokollnachricht zurück:

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

6.3 Die Variable für Status Objekte zurücksetzen

Die Funktion GITFN_STATUSFREE setzt das Status Objekt zurück, was von der Funktion GITFN_STATUS zurückgegeben wurde.
Es wird nur das Parameter: lokaler Repository-Pfad benötigt.

```
data _null_;
    rc=GITFN_STATUSFREE("C:\tmp\git_ksfe2020");
    put rc=;
run;
```

Die Funktionsrückgabe Codes sind:

- 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.

7 Rückgabe von Commit Objekten

7.1 Rückgabe der Commit Log

Die Funktion GITFN_COMMIT_LOG gibt die Anzahl der kommitteten Objekte im lokalen Repository zurück. Die Funktion benötigt nur den Parameter: lokaler Repository Pfad

```
data _null_;
     n = GITFN_COMMIT_LOG("C:\tmp\git_ksfe2020");
     put n=;
run;
```

Der Befehl put n= gibt die Nummer der Commit-Objekte im Repository in der SAS Log zurück.

7.2 Rückgabe der kommitteten Objekten

GITFN_COMMIT_GET gibt ein bestimmtes Attribut von einem kommit Objekt im lokalen Repository zurück. GITFN_COMMIT_LOG muss vorher laufen, um die Commit-Objekte zu erhalten.

```
data _null_;
    length commit_id author_name $200;
    rc=GITFN_COMMIT_GET(1,"C:\tmp\git_ksfe2020", "id", commit_id);
    rc=GITFN_COMMIT_GET(1,"C:\tmp\git_ksfe2020", "author",
author_name);
    put commit_id=;
    put author_name=;
run;
```

GITFN_COMMIT_GET hat vier erforderliche Parameter:

- Anzahl Loops
- lokaler Repository Pfad
- Objekt-Attribut
- Ausgabevariable

Die Funktion GITFN_COMMIT_GET gibt in diesem Beispiel nur ein kommit Objekt zurück, daher ist der Wert 1 für die Variable Anzahl der Loops.

GITFN_COMMIT_GET gibt die folgende Nachricht in der SAS Log zurück:

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

7.3 Die Variable für Commit Objekte zurücksetzen

GITFN_ COMMITFREE setzt das Commit Objekt zurück, das vorher von der Funktion GITFN_COMMIT_GET zurückgegeben wurde. Nur der Parameter: lokaler Repository Pfad ist erforderlich.

```
data _null_;
    rc=GITFN_COMMITFREE("C:\tmp\git_ksfe2020");
    put rc;
run;
```

Die Funktionsrückgabecodes sind:

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

8 Vergleiche zwei Commits

8.1 Rückgabe der Anzahl von Änderungen

Die Funktion GITFN_DIFF gibt die Anzahl der Änderungen zwischen zwei commits zurück und legt ein Diff-Record-Objekt im lokalen Repository an.

```
data _null_;
    n=gitfn_diff(
       "C:\tmp\git_ksfe2020",
       "0d15a34b3a2728480d88166295530d3c2c6b86d2",
       "cf70de661bf032d4c1d0a2c8d32bbe2f366cd15d"
    );
    put n=;
run;
```

GITFN_DIFF hat drei erforderliche Parameter:

- lokaler Repository Pfad
- alte Commit-ID
- neue Commit-ID

Der Befehl put n= gibt die Anzahl der Änderungen zwischen zwei Commits in der SAS Log zurück.

8.2 Rückgabe von Attributen eines Diff-Record Objektes

Die Funktion GITFN_DIFF_GET gibt das spezifizierte Attribut (file, diff_content, diff_type) von einem Diff-Record-Objekt zurück. GITFN_DIFF muss vorher ausgeführt werden um die Commit-Objekte zu erhalten.

```
data _null_;
   length _file_path $ 200;
   rc=gitfn_diff_get(
      1,
      "C:\tmp\git_ksfe2020",
      "0d15a34b3a2728480d88166295530d3c2c6b86d2",
      "cf70de661bf032d4c1d0a2c8d32bbe2f366cd15d",
      "file",
      _file_path
   );
   put _file_path=;
run;
```

GITFN_DIFF_GET hat sechs erforderliche Parameter:

- Anzahl Loops
- lokaler Repository Pfad
- alte Commit-ID
- neue Commit-ID
- Objekt-Attribut
- Ausgabevariable

Der Diff-Inhalt wird in einer JSON-Datei zurückgegeben.

8.3 Zurücksetzen des Diff-Record Objektes

Die Funktion GITFN_ DIFF_FREE löscht den Diff-Datensatz, der von der Funktion GITFN_DIFF zurückgegeben wurde.

```
data _null_;
    rc=gitfn_diff_free(
       "C:\tmp\git_ksfe2020",
       "0d15a34b3a2728480d88166295530d3c2c6b86d2",
       "cf70de661bf032d4c1d0a2c8d32bbe2f366cd15d");
run;
```

GITFN_ DIFF_FREE hat drei erforderliche Parameter:

- lokaler Repository-Pfad
- alte Commit-ID
- neue Commit-ID

Die Funktionsrückgabe Codes sind:

- 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.

8.4 Geänderte Datei vergleichen

Die Funktion GITFN_DIFF_IDX_F gibt Änderungen von Dateien zurück, die sich im Index befinden und von Dateien, die zuletzt in das Repository übertragen wurden. Die Änderungen werden in einer Variablen gespeichert. Die Ausgabe der Diff-Funktionen ist auf 32.767 Zeichen begrenzt, das kann dazu führen, dass die Ausgabe nicht vollständig angezeigt wird.

Die Funktion hat drei erforderliche Parameter:

- lokaler Repository-Pfad
- Dateiname der Dateien, die vergleicht werden sollen
- Variablenname, in der der Inhalt des Diffs gespeichert werden soll

```
data _null_;
    length _diff_content $ 32767;
    _diff_content="";
    rc=GITFN_DIFF_IDX_F("C:\tmp\git_ksfe2020",
        "gitfn_idx_add.sas",
        _diff_content
         );
    put rc=
    put _diff_content=;
run;
```

GITFN_DIFF_IDX_F gibt folgende Nachricht in der SAS-Log zurück:
`NOTE: _diff_content ={ diff-content }`

Die Funktion hat die folgenden Rückgabecodes:

- 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`

Literatur

[1] GitHub Guides: https://guides.github.com/

[2] SAS GIT Dokumentation: https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=n1mlc3f9w9zh9fn13qswiq6hrta0.htm&docsetVersion=9.4&locale=en

[3] 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

[4] An Introduction to GIT Version Control for SAS Programmers by Stephen Philp: https://www.lexjansen.com/wuss/2012/94.pdf