GIT Strategie

Branching & Merging mit GitFlow

LieberLieber Software
Andreas Brauneis
02.07.2019

Inhalt

1. Git Flow

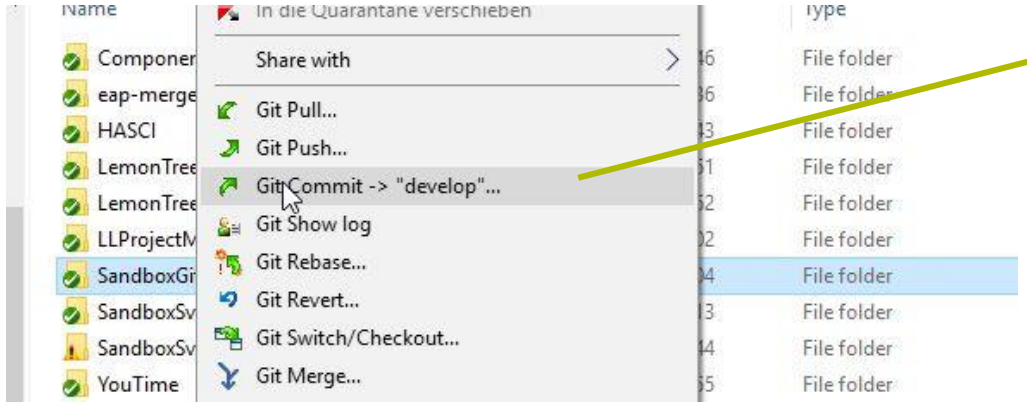
1. GIT Flow: Featurebasiertes Arbeiten

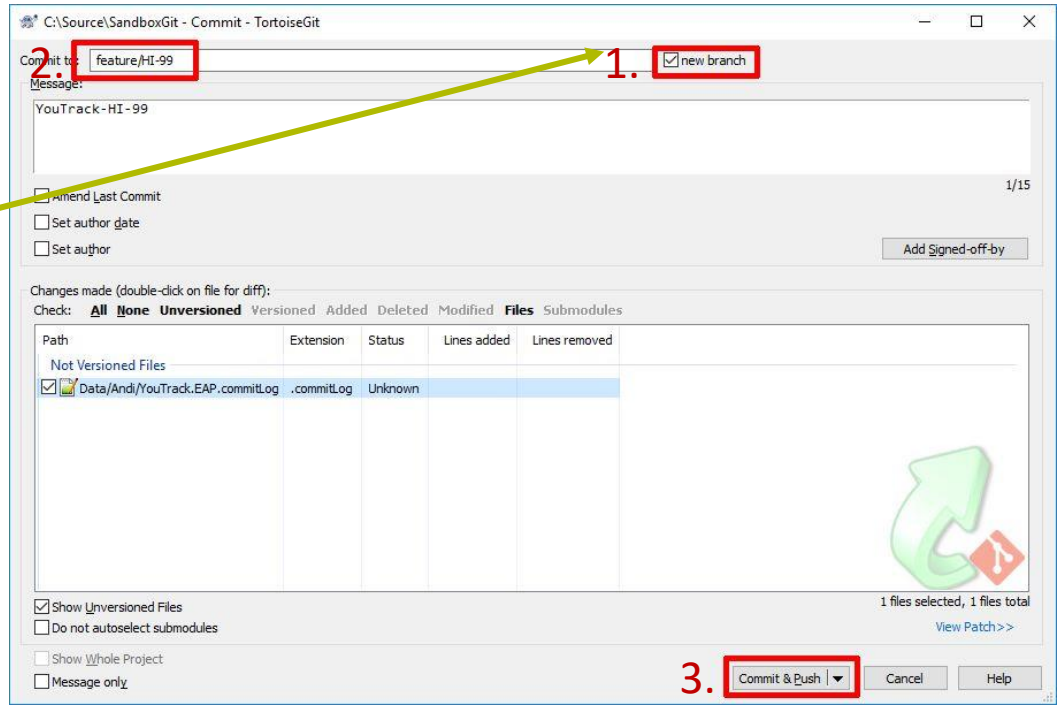
- Es wird nicht direkt in „develop“ gearbeitet.
- Jede Änderung (CR/Feature) wird über einen „Feature-Branch“ eingearbeitet
- Nach getanener Arbeit durch den Modelierer wird der Branch gereviewed und nach „develop“ gebracht.

Es folgt die Beschreibung des Ablaufes anhand des GIT Clients TortoiseGIT

2. Einen neuen Feature-Branch anlegen

Um einen neuen Feature-Branch anzulegen direkt von „Develop“ beim Commit „new branch“ anhaken und unter „Commit to“ den neuen Branch laut naming convention eintragen.
Tortoise-GIT switscht dann automatisch auf diesen Branch und man kann weiterarbeiten und neue Commits anfügen.

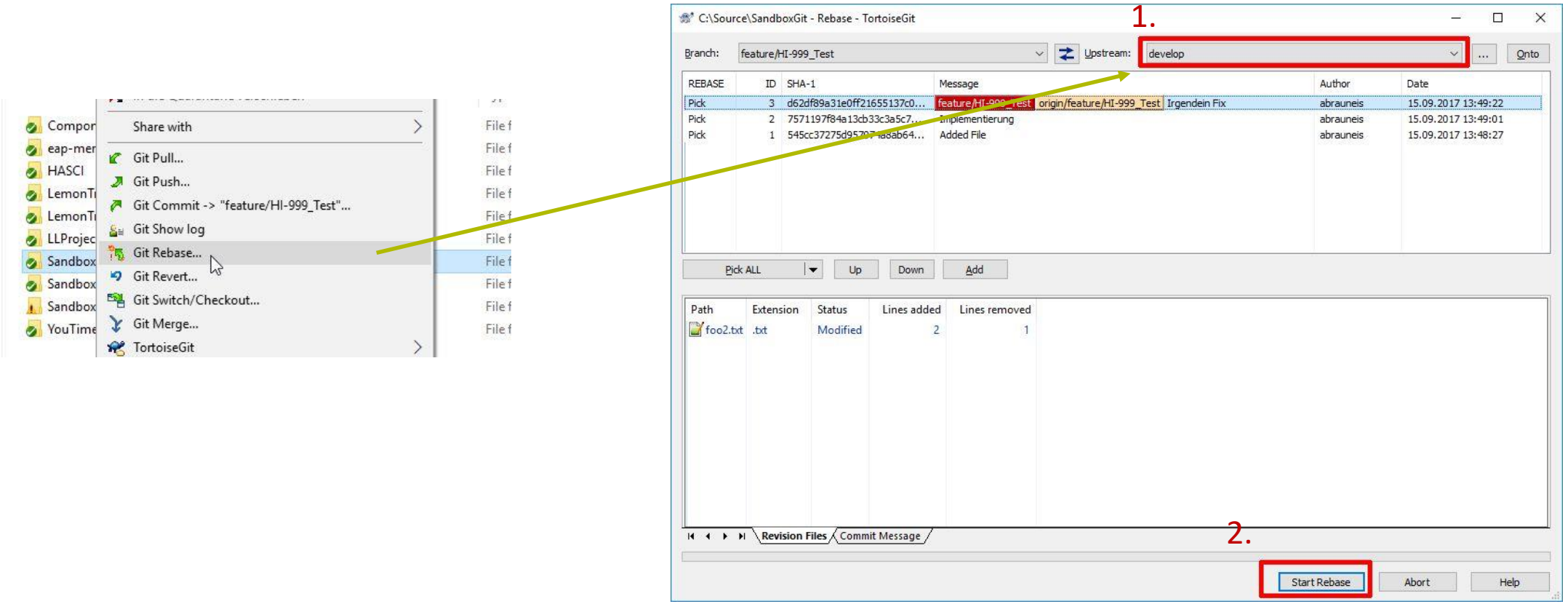




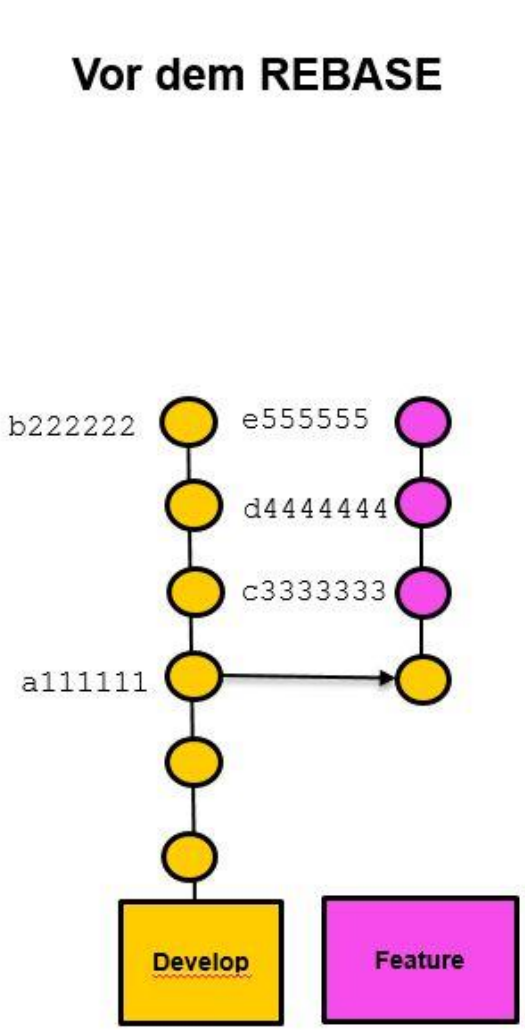
- Mit dem „Commit“ existiert der Branch lokal im Repository.
- Mit dem ersten „Push“ kommt er auch auf den Remote.

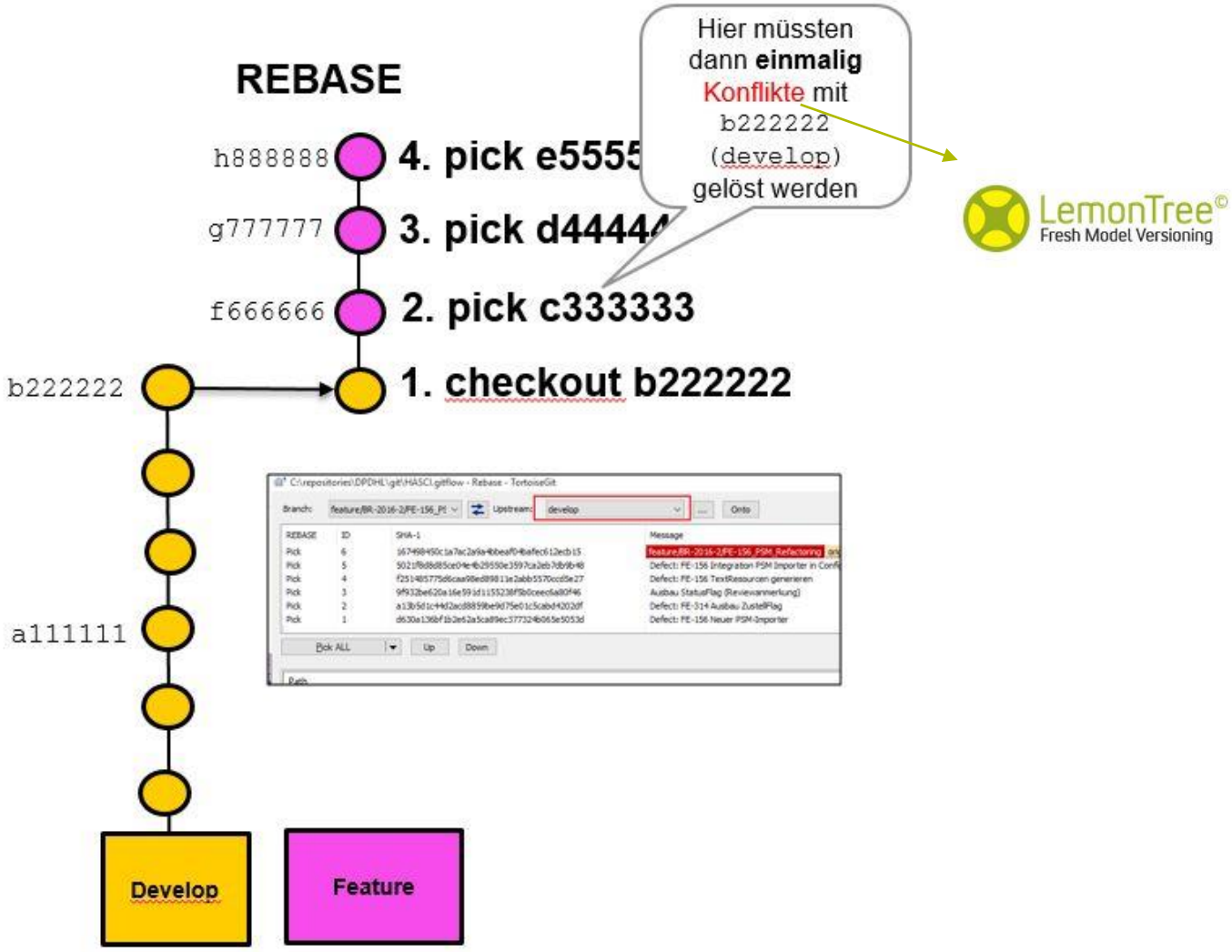
3. Aktualisieren meines Feature Branches aus develop via Rebase

Um den Featurebranch aktuell zu halten diesen auf „Develop“ rebasen. Dies ordnet die neuen Commits des Develops in den Feature Branch VOR die eigenen Commits ein.



Was passiert da?





4. Feature-Branch nach „develop“ bringen

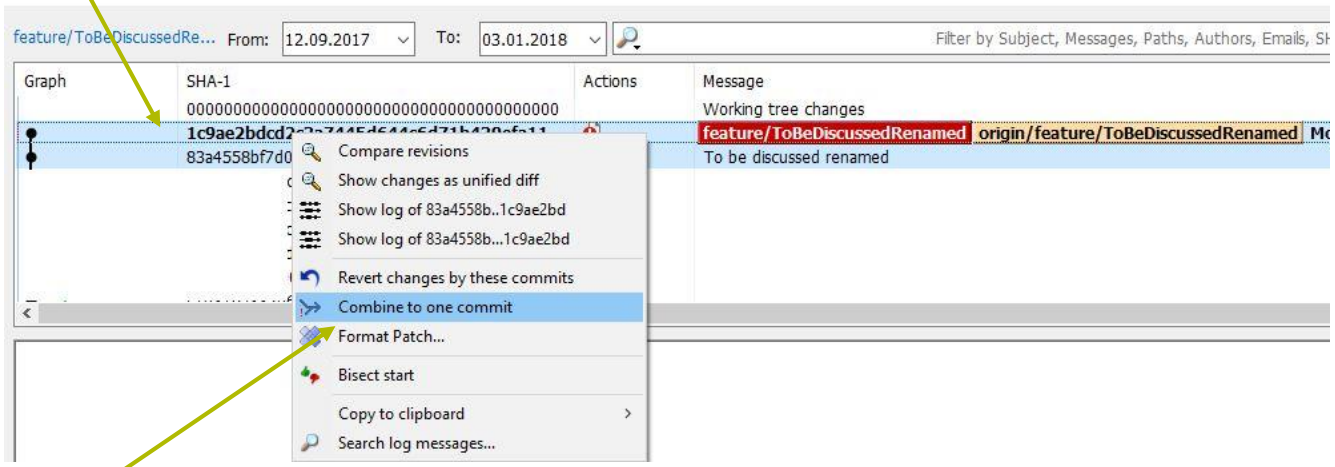
4a Featurebranch für Rückintegration vorbereiten

- Vor dem Zurückmergen des Branches wollen wir diesen komprimieren um eine saubere History zu haben.
- Ziel ist es pro Feature/Task nur EINEN Commit zu haben.
- Dafür werden 2 Schritte benötigt:
  - Schritt 1: Commits kombinieren
  - Schritt 2: Neuen, kombinierten Commit, auf den Remote pushen

4a Featurebranch für Rückintegration vorbereiten

Schritt 1: Commits kombinieren

1.: Im „log“ des Feature Branches werden mehrere aufeinanderfolgende Commits selektiert

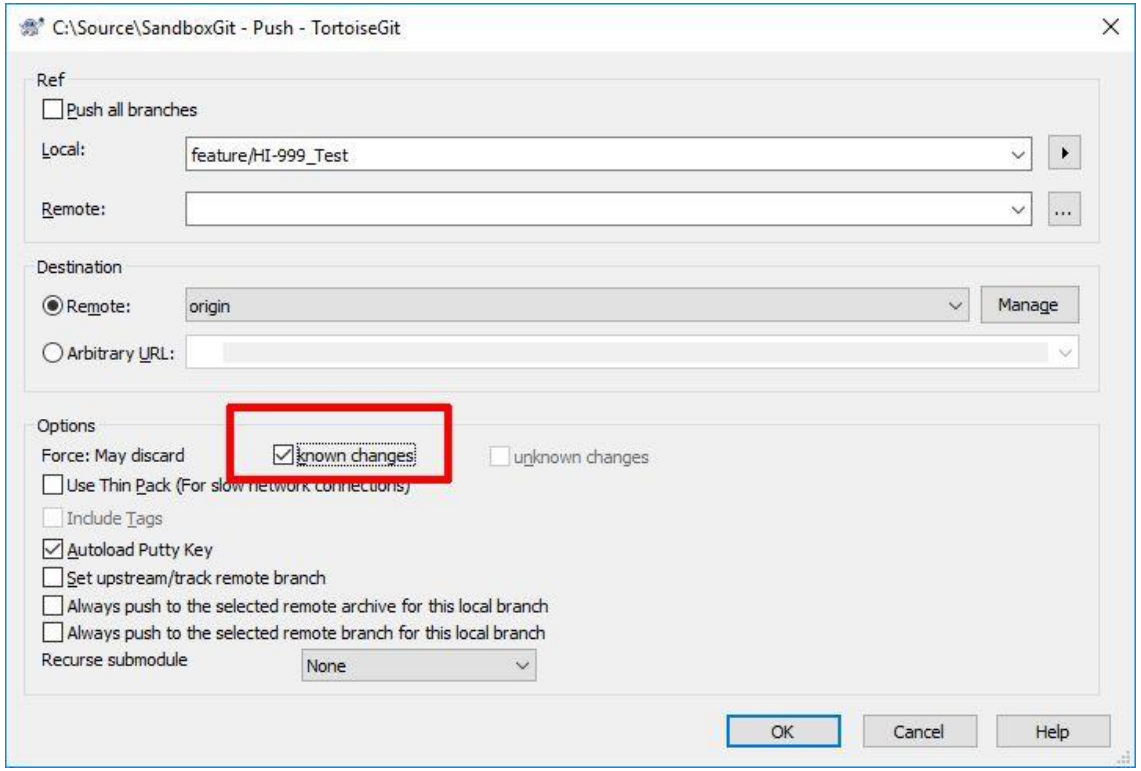


2.: Mittels „Combine to one commit“ werden diese zusammengefügt.

3.: Ein neuer, finaler Commit-Kommentar wird vergeben. Systemvorschlag sollte nicht übernommen werden, da einfach alle Commit-Kommentare übernommen werden. Stattdessen eine sinnvolle Beschreibung der Gesamtänderung bzw. des Tasks.

4a Featurebranch für Rückintegration vorbereiten

Schritt 2: Neuen, kombinierten Commit, auf den Remote pushen



Nach dem Kombinieren wird die Änderung auf den Feature-Branch gepusht. Damit schreiben wir die History des Feature-Branches um und es muss

Force: May discard „known changes“

angehakt werden. Damit weiß GIT dass das unsere Absicht ist. Der Push wird ansonsten abgelehnt.

4b Review durchführen

- Der fertige Featurebranch wird von Kollegen/Reviewer kontrolliert, bevor er nach „develop“ zurückgebracht wird.

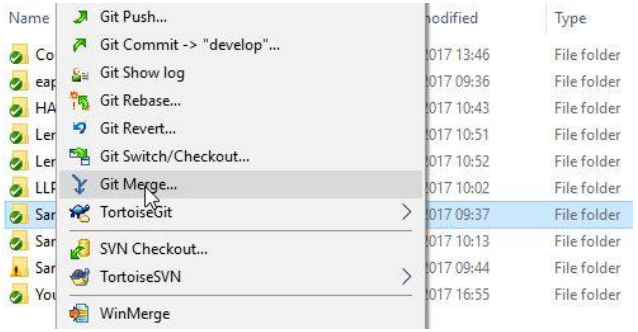
- Wenn Review OK => Weiter mit Schritt 4c => Rückintegration nach „develop“

- Bei Review Anmerkungen => Zurück an Modellierer und nach Einarbeitung der Änderungen wieder mit Schritt 3 starten.

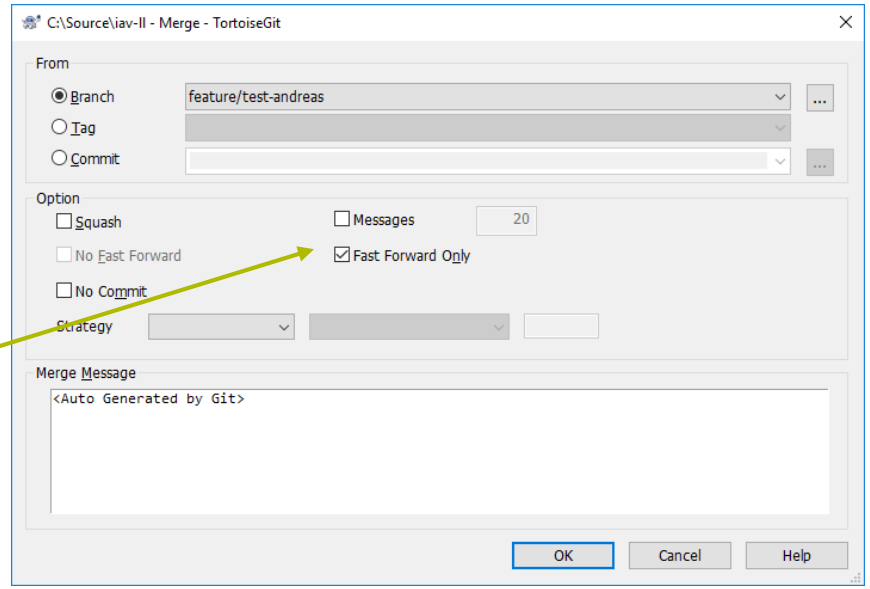
4 Feature-Branch nach „develop“ bringen

4c Feature-Branch zurückmergen

1.: Den Merge-Befehl auswählen (von „develop“)

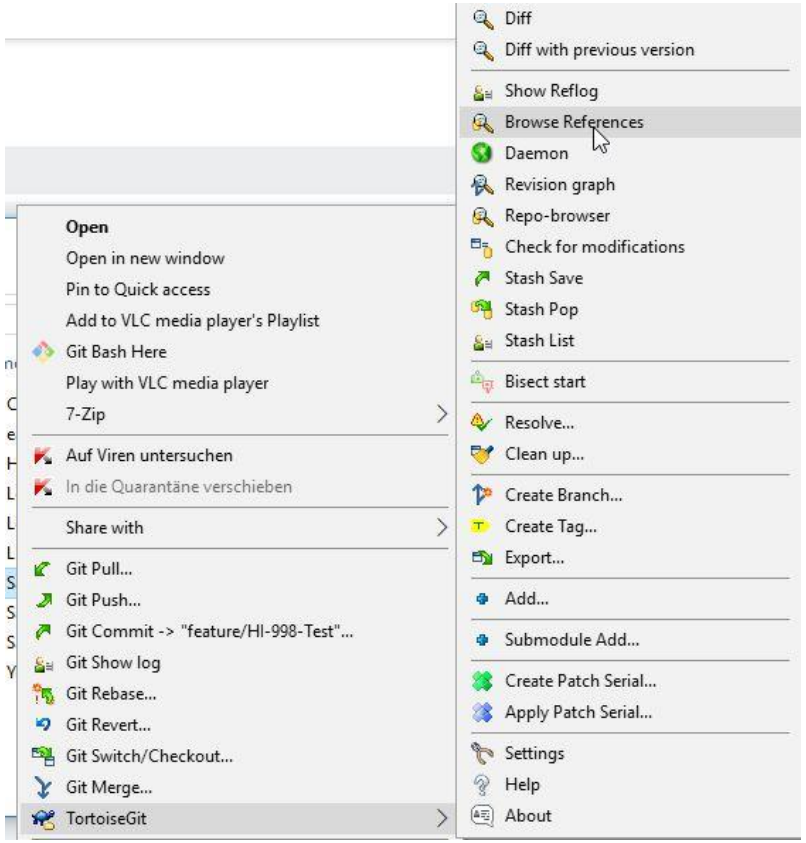


2.: Den Feature-Branch auswählen

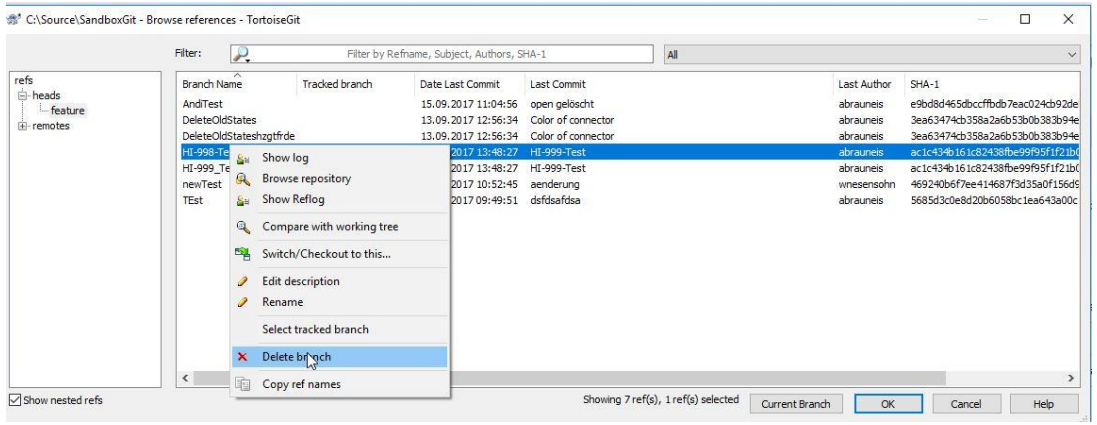


Wichtig ist den Haken bei „Fast Forward Only“ zu setzen. Damit erzeugt GIT keinen Merge-Commit.

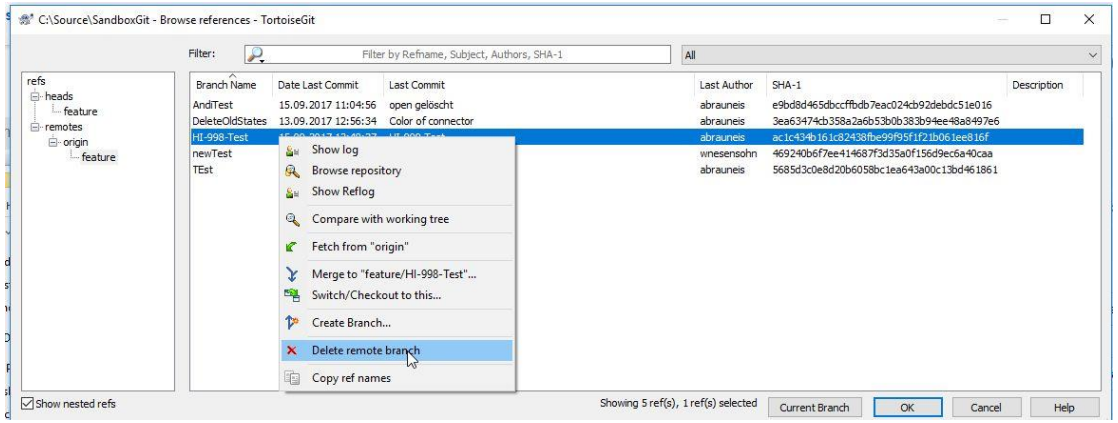
5. Den Feature-Branch lokal und remote löschen



Lokal



Remote



Anmerkungen

- Die Arbeitsschritte wurden mit dem Git-Client „TortoiseGit“ demonstriert. Es können auch andere Clients eingesetzt und auch kombiniert werden (SmartGIT, GitLab, SourceTree, …).
- Beim Verwenden des Workflows „GitFlow“ ist der „develop“-Branch immer „stabil“. Es sind nur vollständig modellierte Features/Change Requests integriert.
- Semantic Versioning ist möglich: GitVersion.
- Workflow kommt aus der Softwareentwicklung und ist schon jahrelang erprobt.
- Bei LieberLieber wird der Workflow in der Softwareentwicklung und beim gemeinsamen Modelieren verwendet.

Appendix

- Buch über GIT: https://git-scm.com/book/en/v2
- GitVersion: https://github.com/GitTools/GitVersion
- GitFlow: https://datasift.github.io/gitflow/IntroducingGitFlow.html