

Tutorial – Hochladen von Dateien in das PC/GEOS-Repository

Dieses Tutorial beschreibt, wie man Dateien (hier am Beispiel übersetzter Hilfe-Dateien) auf github hochladen kann, so dass sie danach für alle automatisch zur Verfügung stehen. Der hier beschriebene Weg ist von mir erprobt. Möglicherweise (sogar sehr wahrscheinlich) gibt es weitere Möglichkeiten.

Das Tutorial geht zunächst davon aus, dass Windows benutzt wird. Für Linux sollten die Schritte analog funktionieren – falls nicht, bitte melden!

Das hier beschriebene Verfahren funktioniert prinzipiell für alle Dateien, die man PC/GEOS hinzufügen will oder die man ändern / anpassen will, also z.B. Code-Dateien oder Übersetzungsdateien für Programme oder Libraries.

Wenn man, wie hier vorausgesetzt, nur Daten-Dateien (z.B. Hilfe-Dateien) hochladen will, muss man nicht das SDK „zum Laufen“ bringen. Man braucht also weder den Watcom-Compiler oder perl, noch muss man irgendwelche Pfade setzen.

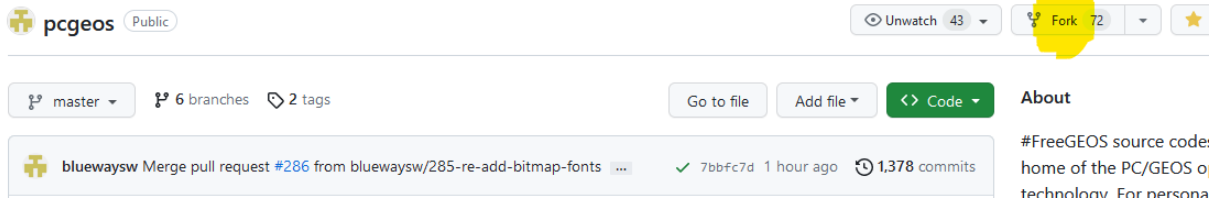
Noch ein Hinweis: Bei allem was wir machen, müssen wir strikt die Groß/Kleinschreibung beachten. Paul.txt und PAUL.TXT sind für git verschiedene Dateien, für Windows jedoch nicht. Also nicht „mal eben“ die Groß/Kleinschreibung eines Dateinamens ändern, das kann richtig ins Auge gehen.

Das Prinzip

1. Man benötigt eine lokale Kopie der Dateien von Github („Repository“)
2. In dieser lokalen Kopie legt man die geänderten Dateien an der der dafür vorgesehenen Stelle ab.
3. Man lädt die Änderungen auf Github hoch.
4. Man macht einen „pull request“ – also eine Anforderung an Falk, die Änderungen in das originale Repository zu übernehmen.
5. Falk akzeptiert die Änderungen. Danach sind die geänderten Dateien im originalen github Repository und jeder, der seine lokale Kopie updatet, bekommt sie.

Schritt 0: Vorbereitungen

- Herunterladen und installieren von GIT (<https://git-scm.com/downloads>)
Git ist ein „Versions-Kontroll-Programm“, das lokal auf unserem Rechner läuft. Es protokolliert alle Änderungen in unserem Repository und ermöglicht uns, geänderte Dateien auf github hoch- oder von dort herunter zu laden.
- Optional: Herunterladen einer Explorer-Erweiterung für Git. Auf Anraten von Falk benutze ich TortoiseGit. Damit kann man alle unten beschriebenen Schritte statt auf der Kommandozeile komfortabel aus dem Explorer heraus ausführen und bestimmte Sachen funktionieren automatisch. Für Linux gibt es sicher analoge Lösungen.
- Erstellen eines eigenen Accounts auf Github (<https://github.com>)
Für dieses Tutorial nehmen wir an, dein Benutzername sei **geo-paulchen**. Diesen Benutzer habe ich extra für dieses Tutorial erstellt.
- Erstellen einer eigenen Kopie („Fork“) des originalen Repositories
Über diese Kopie wird der Transfer der geänderten Dateien zwischen dem originalen Repository auf github und dem lokalen Repository auf dem eigenen Rechner abgewickelt.
 - Melde dich auf Github an (Erinnerung: du bist hier geo-paulchen)
 - Gehe auf <https://github.com/bluewaysw/pcgeos>
 - Klicke auf „Fork“
Eine Kopie wird erstellt. Sie ist unter <https://github.com/geo-paulchen/pcgeos> erreichbar.



Für dieses Tutorial benötigen wir einige wenige git-Kommandos, die wir auf der Kommandozeile eingeben. In den folgenden Schritten wird TortoiseGit oder ähnliches nicht mehr explizit erwähnt. Für alle Kommandozeilen-Befehle gibt es entsprechende Einträge im Kontext-Menü.

git clone

Erstellt eine lokale Kopie eines Repository von github.

git status

Zeigt wichtige Informationen an, z.B. ob es geänderte oder neue Dateien gibt.

git add

Markiert (neue oder geänderte) Dateien als „fertig“.

Hintergrund: Im git-Jargon heißt das „in den staging Bereich verschieben“. Die Datei wird natürlich nicht wirklich verschoben, sondern nur in die Liste der „fertigen“ Dateien aufgenommen.

git commit

Bereitet die „fertigen“ Dateien zum Hochladen vor.

Der interne Zweck dieses Schritts ist das Erstellen eines „Schnappschusses“ des aktuellen Entwicklungsstandes, quasi einen „Wiederherstellungspunkt“, zu dem man jederzeit zurückkehren kann. Das passiert sehr effizient, da nur die Unterschiede zum vorherigen Zustand gespeichert werden.

git push

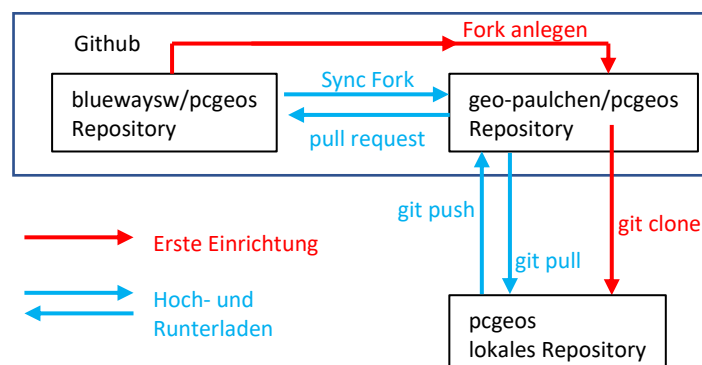
Lädt die committeten Dateien in das eigene (!) Repository hoch.

git pull

Lädt Änderungen, die am zentralen Repository (bluewaysw/pcgeos) gemacht wurden, in das eigene lokale Repository herunter.

Dazu muss man vorher sein eigenes Repository auf der github Webseite mit den Änderungen des zentralen bluewaysw Repository synchronisieren.

Der prinzipielle Arbeitsablauf mit eine Repository und git lässt sich durch folgendes Bild beschreiben.



Schritt 1: Erstellen einer lokalen Kopie des PC/GEOS Repositories

Wir erstellen eine Kopie des gerade neu erstellten Forks.

- Wähle einen (leeren) Ordner auf deiner Festplatte, z.B. C:\Tutorial
- Starte die Kommandoebene (Windows: cmd.exe) und wechsle in den oben gewählten Ordner
- Gib ein

git clone https://github.com/geo-paulchen/pcgeos

Natürlich musst du geo-paulchen durch deinen wirklichen Benutzernamen ersetzen.

In C:\Tutorial\pcgeos findest du jetzt eine Kopie deines Repository – das natürlich (noch) mit dem von bluewaysw übereinstimmt. Mit dem Befehl **git status** können wir uns anzeigen lassen, ob alles in Ordnung ist.

```
C:\Tutorial>git clone https://github.com/geo-paulchen/pcgeos
Cloning into 'pcgeos'...
remote: Enumerating objects: 41500, done.
remote: Counting objects: 100% (6305/6305), done.
remote: Compressing objects: 100% (2577/2577), done.
remote: Total 41500 (delta 3945), reused 5711 (delta 3487), pack-reused 35195
Receiving objects: 100% (41500/41500), 99.88 MiB | 8.54 MiB/s, done.
Resolving deltas: 100% (17559/17559), done.
Updating files: 100% (23263/23263), done.

C:\Tutorial>cd pcgeos

C:\Tutorial\pcgeos>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

C:\Tutorial\pcgeos>
```

Hinweis: Wie können jedes x-beliebige Repository von github clonen. Dateien mit git push hochladen, können wir jedoch nur in unsere eigenen Repositories, in unserem Fall also in unseren Fork des bluewaysw/pcgeos Repositories.

Schritt 2: Ablegen der geänderten Dateien

Hier ist wirklich nichts weiter zu tun, als die geänderten Dateien an die vorgesehene Stelle im Repository zu kopieren.

- Die Hilfequellen (GeoWrite Dateien) der deutschen Übersetzung gehören nach pcgeos\Tools\build\product\bbxensem\HelpSource\german
- Die übersetzten Hilfe-Dateien (aus USERDATA\HELP) gehören nach pcgeos\Tools\build\product\bbxensem\Help\german
- Übersetzungsdateien für Programme und Libraries kommen nach pcgeos\Tools\build\product\bbxensem\Trans\german

Man kann auch neue Dateien hinzufügen. Es ist vereinbart, dass die Dateinamen in den genannten Ordnern in Großbuchstaben sind.

Abschließend sollten wir prüfen, ob alles OK ist. Wechsle auf der Kommandoebene nach pcgeos (oder einem seiner Unterverzeichnisse) und gib ein

git status

Im folgenden Beispiel habe ich eine Datei geändert (ARTIST.000). Sie erscheint unter „Changes not staged for commit“. Außerdem habe ich eine Datei hinzugefügt (PAULCHEN.000). Sie erscheint unter „Untracked files“, weil git sie noch nicht kennt.

```
C:\Tutorial\pcgeos>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Tools/build/product/bbxensem/HelpSource/ARTIST.000

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Tools/build/product/bbxensem/HelpSource/PAULCHEN.000

no changes added to commit (use "git add" and/or "git commit -a")

C:\Tutorial\pcgeos>
```

Wichtiger Hinweis: Falls du eine Datei **umbenennen** möchtest, **musst** du das mit dem git Befehl **git mv** machen (oder mit TortoiseGit o.ä.), **niemals** einfach mit dem Explorer. Git kann sonst kräftig durcheinander kommen.

git mv PAULCHEN.000 NEWNAME.000

Anmerkung: Git ist keine Mimose, in vielen Fällen funktioniert das Umbenennen mit dem Explorer oder ähnlichem ohne Probleme. Aber nicht in allen, insbesondere dann nicht, wenn die Groß/Kleinschreibung geändert wird. Deshalb: einfach kein Risiko eingehen und **git mv** benutzen.

Schritt 3: Vorbereiten zum Hochladen

Da man nicht immer alle Änderungen, die man gemacht hat, hochladen will, erfolgt das Hochladen in mehreren Schritten:

- Hinzufügen der Dateien zur Liste der fertigen Dateien mit ‚git add‘
- Liste zum Hochladen vorbereiten mit ‚git commit‘
- Dateien hochladen mit ‚git push‘

- Wechsle in den Ordner, der die geänderten Dateien enthält, und gib ein

git add .

Der Punkt hinter git add bewirkt, dass alle geänderten und neuen Dateien in diesem Ordner *und seinen Unterordnern* zur Liste der „fertigen“ Dateien hinzugefügt werden. Will man nur einzelne Dateien hinzufügen, kann man auch ‚git add filename‘ verwenden.

Um zu prüfen, ob wie erfolgreich waren, verwenden wir ‚git status‘.

```
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   ARTIST.000
        new file:   PAULCHEN.000

C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>
```

Alle Dateien, die wir hochladen wollen, müssen jetzt unter „Changes to be committed“ erscheinen.

- Hinweis: Wenn wir eine der Dateien anschließend noch ändern, werden sie aus „Fertig“-Liste entfernt, d.h sie erscheinen bei ‚git status‘ wieder in rot als geändert. Wir müssen sie dann erneut adden.

- Gib ein

git commit -m "German help files for artist"

-m steht für message und der Text in Anführungsstrichen ist ein Kommentar, der beschreiben soll, welche Änderungen durchgeführt wurden.

```
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git commit -m "German help files for artist"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'rainerb@PHYSIK-ABI.(none)')
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>
```

Ups. Beim ersten Mal weiß git noch nicht, wer wir sind und damit, ob wir das Recht haben in unser Repository zu schreiben.

Na dann machen wir doch einfach, was git vorschlägt. Vor dem global stehen zwei Minuszeichen. Anschließend klappt das auch mit dem Commit.

```
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git config --global user.email "rabesoft@freenet.de"
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git config --global user.name "geo-paulchen"
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git commit -m "German help files for artist"
[master 5ef03533] German help files for artist
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Tools/build/product/bbxensem/HelpSource/PAULCHEN.000
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>
```

Um den Erfolg zu prüfen, verwenden wir zunächst den Befehl ‚git log‘.

```
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git log
commit 5ef035330b2c1db9d138e477b9702a4c00c34b8c (HEAD -> master)
Author: geo-paulchen <rabesoft@freenet.de>
Date: Tue Jan 16 13:20:35 2024 +0100

    German help files for artist

commit aa62d6fec1ab74f031adc0268390ac69c3c7d082 (origin/master, origin/HEAD)
Merge: 94ecc5a8 3333b3d5
Author: blueway.Softworks <45116648+bluewaysw@users.noreply.github.com>
Date: Sun Jan 14 15:47:14 2024 +0100

    Merge pull request #381 from nschulux/master

    Gourmet updated code for localization.

commit 3333b3d5279a500305df8529c1b4da8aee537a23
Author: Nico <118665816+nschulux@users.noreply.github.com>
Date: Sun Jan 14 09:46:33 2024 +0100
```

‚git log‘ zeigt die Liste aller bisherigen commit-Befehle (kurz ‚commits‘) an, einschließlich des dazugehörigen Kommentars sowie wann und von wem sie gemacht wurden.

Wichtig! Um wieder auf die Kommandozeile zurück zu kommen, drückt man die Taste **q**.

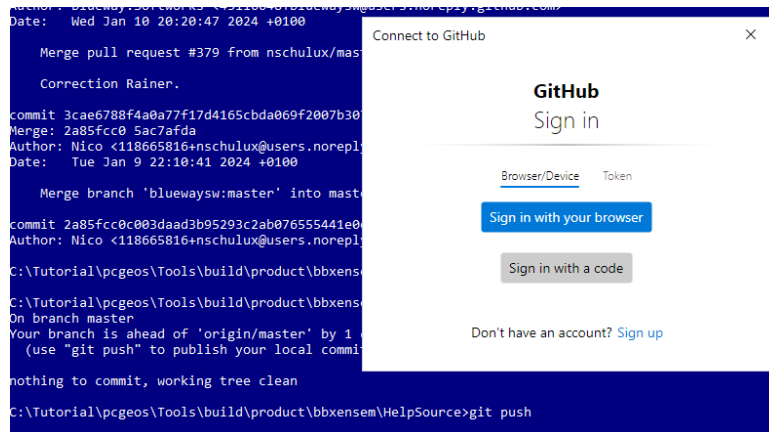
Mit ‚git status‘ können wir nun prüfen, ob noch ungeänderte Dateien vorhanden sind, die wir eventuell übersehen haben. Hier im Beispiel sehen wir, dass wir einen commit gemacht haben, der noch nicht hochgeladen (gepusht) ist.

```
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
C:\Tutorial\pcgeos\Tools\build\product\bbxensem\HelpSource>
```

Schritt 4: Hochladen der geänderten / neuen Dateien.

- Wenn alles Ok ist können wir eingeben
git push
Git lädt die geänderten Dateien in unser Repository auf github hoch.

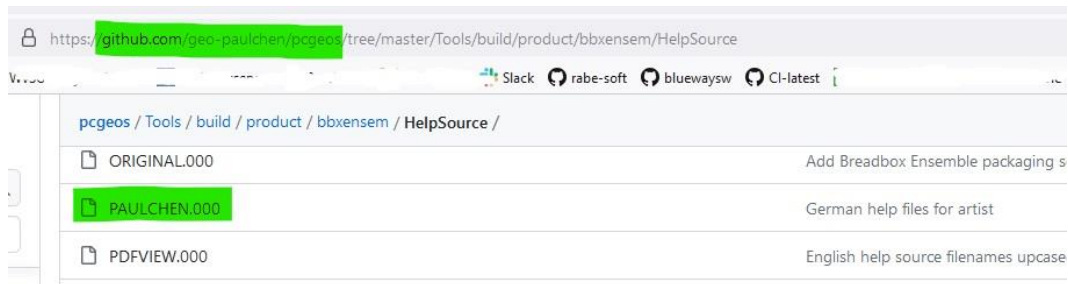


Nächstes Ups. Wir müssen uns bei github anmelden, damit github unseren push (Schreiben von Dateien in das Repository) akzeptiert. Ich habe „Sign in with your browser“ gewählt, der Browser öffnet sich und man muss die dortigen Anweisungen befolgen (akzeptieren).

Anmerkung: Auf dem Rechner, auf dem ich vorher TortoiseGit installiert hatte, erschien diese Aufforderung nicht. Dort habe ich die Daten bei der Installation von TortoiseGit eingegeben.

```
C:\Tutorial\pcgeos>git push
info: please complete authentication in your browser...
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 4.04 KiB | 689.00 KiB/s, done.
Total 9 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com:geo-paulchen/pcgeos
   aa62d6fe..5ef03533  master -> master
C:\Tutorial\pcgeos>
```

Danach läuft der Prozess automatisch weiter. Die Dateien sind jetzt in deinem Repository auf Github verfügbar.



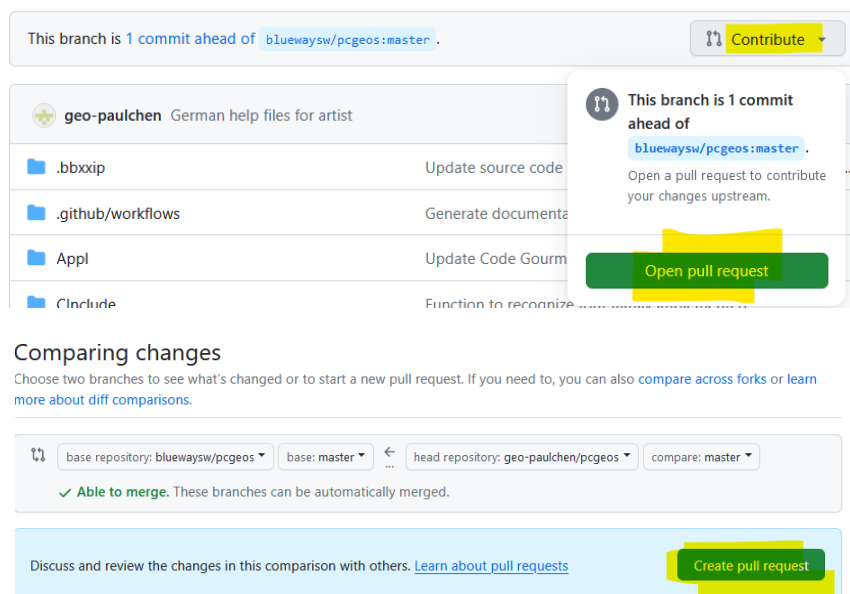
Die gute Nachricht ist: Nachdem wir uns einmal authentifiziert haben wissen sowohl git (lokal) als auch github (im Web), wer wir sind. Ein git push läuft dann ohne weitere Nachfragen durch.

Schritt 5: Einen ‚pull request‘ machen

Ein pull request ist eine Bitte (request = Anforderung) an den Besitzer des Repositories, sich die Änderungen in das originale Repository zu ziehen (pull).

Das müssen wir direkt auf der github Webseite machen.

- Gehe auf <https://github.com/geo-paulchen/pcgeos> und melde dich an.
Natürlich musst du hier wieder geo-paulchen durch deinen eigenen Accountnamen ersetzen.
- Klicke auf „Contribute“ und dann auf „Open pull request“

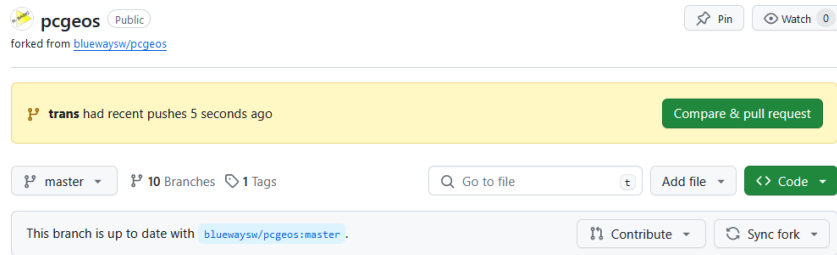


An dieser Stelle kannst du noch einmal vergleichen, welche Dateien in den pull request aufgenommen werden. Bei Codefiles (reinen Textdateien) kann man auch die Änderungen zur vorherigen Version ansehen.

- Klicke dann auf „Create pull request“

An dieser Stelle wird die Anforderung (pull request) an das originale Repository weitergeleitet. Ich führe das für das Tutorial jetzt nicht aus, da die Datei PAULCHEN.000 nur zu Demonstrationszwecken hochgeladen wurde und nicht wirklich in das originale Repository aufgenommen werden soll.

Nachdem der erste pull request durch ist, sollte dir nach jedem ‚git push‘ direkt ein pull request angeboten werden, ohne den Umweg über das Contribute-Menü.



Schritt 6: Die Änderungen werden akzeptiert.

Jetzt heißt es warten, bis Falk die Änderungen übernommen hat. Zwischendurch kannst du weitere Dateien übersetzen oder ändern, diese committen und pushen. Diese werden, solange der pull request noch nicht akzeptiert ist, dem aktuellen pull request zugeschlagen. Falls nicht, muss du einen weiteren pull request machen.

Up-To-Date bleiben

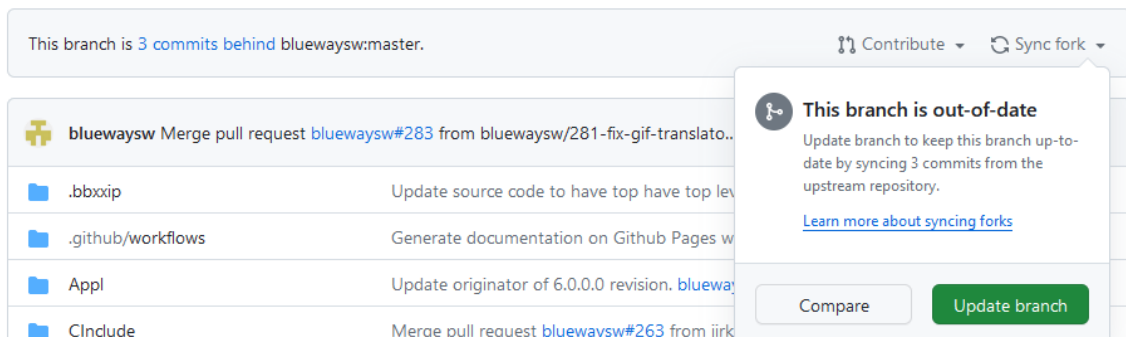
Wenn Falk das originale Repository geändert hat, z.B. weil jemand anders einen pull request gemacht hat, müssen wir unser eigenes Repository updaten.

Das passiert in zwei Schritten:

1. Gehe auf <https://github.com/geo-paulchen/pcgeos> und melde dich an.

Natürlich musst du hier wieder geo-paulchen durch deinen eigenen Accountnamen ersetzen.

- Klicke auf „Sync fork“ und dann auf „Update branch“.



2. Wechsle auf deinem Rechner in dein lokales Repository (z.B. C:\Tutorial\pcgeos) und gib ein **git pull**

Git updated dein lokales Repository.

In seltenen Fällen gibt es hier eine Meldung über einen „Konflikt“. Typischer Weise haben du und jemand anders die gleiche Datei geändert. Solche Konflikte sind immer lösbar, das soll jedoch nicht Inhalt des Tutorials.

Hilfe, ich hab' Mist gebaut!

Keine Panik, git ist ein anderer Name für „Mist reparieren“. Hier die häufigsten Fehler und wie man sie repariert. Denke bitte daran, dass die Groß/Kleinschreibung, auch von Dateinamen unter Windows, wichtig ist.

Natürlich kann git nicht Hellsehen. Jedes „Zurücknehmen“ setzt die betroffenen Dateien auf den Stand des letzten commit zurück. Nicht umsonst habe ich oben von „Wiederherstellungspunkt“ gesprochen.

- Ich habe aus Versehen eine Datei gelöscht.
git restore <filename>
- Ich habe aus Versehen eine Datei geändert, die ich gar nicht ändern wollte.
git restore <filename>
- Ich habe versehentlich eine Datei geaddet, (git add), die ich gar nicht hochladen will.
→ Frag mal ‚git status‘ um Rat. Dann siehst du:
git restore --staged <filename>
Vor dem staged sind zwei Minuszeichen.
- Ich habe eine Datei als fertig angesehen und geaddt, aber es gibt noch etwas zu korrigieren.
→ korrigiere die Datei und add sie dann erneut.
- Ich habe nach einem Commit festgestellt, dass eine Datei noch einen Fehler enthält.
→ Korrigiere die Datei, add sie und committe sie erneut. Du hast dann zwei (oder sogar noch mehr) commits. Das macht nichts, beim nächsten ‚git push‘ werden alle noch nicht gepushten commits hochgeladen.
- Es ist noch etwas zu korrigieren (z.B. Rechtschreibfehler), aber ich habe die Datei schon gepusht und einen pull request gemacht.
→ Kein Problem. Ändere die Datei, add sie, committe und pushe die Änderungen. Dann schaust du auf github, ob ein erneuter pull request erforderlich ist.
Das funktioniert sogar, wenn Falk die alte (fehlerhafte) Datei bereits akzeptiert hat.
- Ich habe eine Datei in einen pull request aufgenommen, die gar nicht hochgeladen werden sollte.
→ Auch das ist kein Problem.
 1. Mach eine Sicherheitskopie der Datei.
 2. Lösche die Datei.
 3. Verwende „git add .“ um die Information, dass eine Datei gelöscht wurde, in die Liste der zu commitenden Änderungen aufzunehmen.
 4. Committe und pushe die Änderungen. Dein pull request wird geändert, die Datei ist nicht mehr Teil des pull request.
Falls der pull request bereits akzeptiert wurde musst du einen neuen pull request machen, damit der Löschvorgang in das originale Repository übernommen wird.
 5. Falls sinnvoll: Stelle die Datei aus der Sicherheitskopie wieder her.
- Ich habe ein anderes Problem, das hier nicht aufgeführt ist.
→ Suche im Internet. Häufig findest du dort eine Lösung.
→ Frage jemandem der sich mit git auskennt.