



Technische  
Universität  
Braunschweig

Institut für  
Flugführung



# Versionskontrolle mit GITHUB

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,  
30. April 2019

# Agenda

- 09. April Einführung
- 16. April Softwareprojektmanagement
- 23. April Entwicklungstools

## **30. April GitHub**

- 07. Mai Dokumentation und Bug-Reporting
- 14. Mai Einführung Arduino
- 21. Mai Dateieingabe und -ausgabe
- 28. Mai Einführung von Qt

## 4. & 11. Juni Tag der Lehre und Exkursionswoche

- 18. Juni GUI-Erstellung mit Qt
- 25. Juni Serielle Kommunikation
- 02. Juli API-Anleitung und Projektarbeit
- 09. Juli Vorbereitung der Abgabe
- 16. Juli Fragen

12. August 10:00 Abgabe

## Versionskontrolle mit GITHUB

Als Teilnehmer soll ich am Ende dieser Übung...

- ☐ Konflikte in git lösen können
- ☐ an bestehenden Projekten weiterarbeiten können
- ☐ das lokale Repository mit GITHUB synchronisieren können
- ☐ das Projekt-Wiki erstellen und mit Inhalt füllen können

# git-Grundlagen



## Konflikte

# Was sind Konflikte?

Konflikte ...

- sind gleichzeitige Änderungen in gleichen Abschnitten einer Datei
  1. in unterschiedlichen Zweigen, oder
  2. von unterschiedlichen Nutzern im gleichen Zweig
- tauchen beim Committen sowie Zusammenführen von Zweigen auf
- müssen händisch aufgelöst werden

Kommunikation mit anderen Teammitgliedern erforderlich!

# Beispiel

# Entstehung eines Konfliktes

demo.txt wird in zwei Zweigen in der selben Zeile bearbeitet.  
→ Konflikt beim Merge.

## MASTER-BRANCH

```
1 #demo
2
3 Hello World
4 Oder auf deutsch: "Hallo Welt". Aber warum schreiben
   Programmierer eigentlich so oft "Hello World"?
```

## ENGLISH-BRANCH

```
1 #demo
2
3 Hello World
4 Why do programmers love greeting the world so much?
```

# Darstellung von Konflikten



git fügt an der Stelle des Konfliktes eine Sequenz entsprechend des untenstehenden Schemas ein.

## INHALT VON README.MD

```
1 # demo
2
3 Hello World
4 <<<<<< HEAD
5 Oder auf deutsch: "Hallo Welt". Aber warum schreiben
   Programmierer eigentlich so oft "Hello World"?
6 =====
7 Why do programmers love greeting the world so much?
8 >>>>>> english
```

# Behebung von Konflikten



Die betroffene Datei kann manuell bearbeitet werden.

NEUER INHALT VON DEMO.TXT

```
1 # demo
2
3 Hello World
4 Oder auf deutsch: "Hallo Welt". Aber warum lieben
   Programmierer es eigentlich so sehr, die Welt zu grüßen?
```

## Hinweise

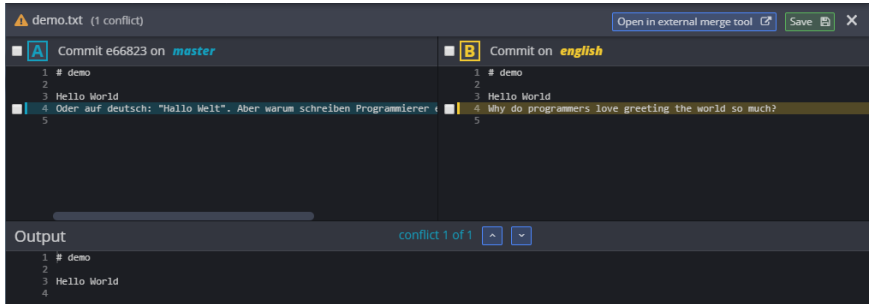
1. Die Konfliktmarkierungen müssen entfernt werden.
2. Es muss nicht zwangsläufig eine der beiden Konfliktversionen verwendet werden. Eine Kombination ist auch möglich (siehe oben).





# Behebung von Konflikten

Alternativ kann ein Diff-Tool (z.B. GITKRAKEN) verwendet werden

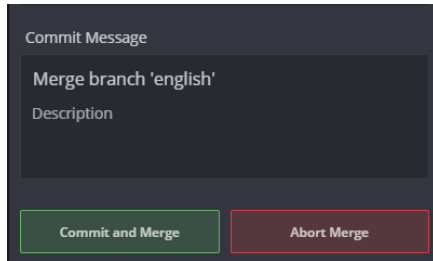


- Die unterschiedlichen Versionen werden gegenübergestellt
- Es können Blöcke oder einzelne Zeilen ausgewählt und modifiziert werden

# Behebung von Konflikten

## COMMIT

Nachdem alle Konflikte beseitigt wurden, muss die neue „zusammengeführte“ Version mit einem `Commit` im Repository gesichert werden.



Commit Message

Merge branch 'english'

Description

Commit and Merge

Abort Merge

# Aufgaben

1. Ein Repository anlegen
2. Die Datei `demo.txt` anlegen und mit 4 Zeilen Beispielttext speichern
3. `demo.txt` über ein Commit im Repository sichern
4. Einen neuen Zweig erstellen
5. `demo.txt` Inhalt von Zeile 2 bearbeiten
6. In den master-Zweig wechseln
7. `demo.txt` Inhalt von Zeile 2 bearbeiten
8. Den erstellten Zweig in den master-Zweig überführen
9. Konflikt auflösen
10. Zusammengeführte Version über ein Commit speichern

Gibt es Fragen oder Anmerkungen zu dem Thema  
**git-Grundlagen?**



# Abgehakt

## Versionskontrolle mit GITHUB

Als Teilnehmer soll ich am Ende dieser Übung...



- ☒ Konflikte in git lösen können
- ☐ an bestehenden Projekten weiterarbeiten können
- ☐ das lokale Repository mit GITHUB synchronisieren können
- ☐ das Projekt-Wiki erstellen und mit Inhalt füllen können

Teamarbeit  
mit git



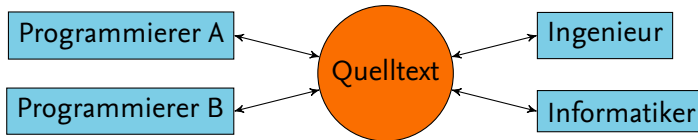
Teamarbeit

Repo-  
Varianten

# Softwareentwicklung im Team

## Ausgangsszenario

- Implementierung einer Steuerungssoftware
- Softwareentwicklung im Team
  - Programmierer
  - Ingenieure
  - Informatiker
- Arbeit an gemeinsamer Quelltextbasis



# Varianten von Versionskontrollsystemen

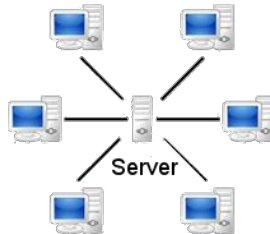
## Lokal

Revisionen werden im lokalen Dateisystem gespeichert



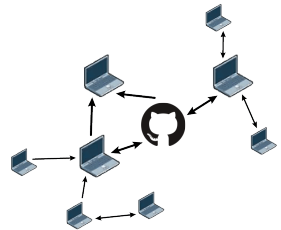
## Zentral

Revisionen werden zentral auf einem Server verwaltet



## Verteilt

Jede Instanz ist eine Kopie des gesamten Repositories





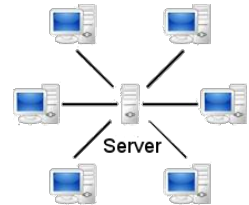
# Lokale Versionskontrollsysteme

- Einfachste Möglichkeit des Versionskontrollsystems
- Änderungen und Verwaltungsinformationen werden lokal gespeichert
  - Kein geregelter Austausch zwischen Entwicklern
  - In der Regel nur Verwaltung einzelner Dateien
  - Überblick über gemachte Änderungen dennoch möglich
  - Verwendung heutzutage z. B. für Konfigurationsdateien



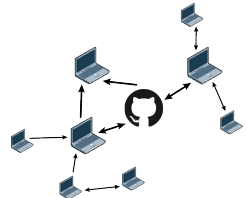
# Zentrale Versionskontrollsysteme: Funktion

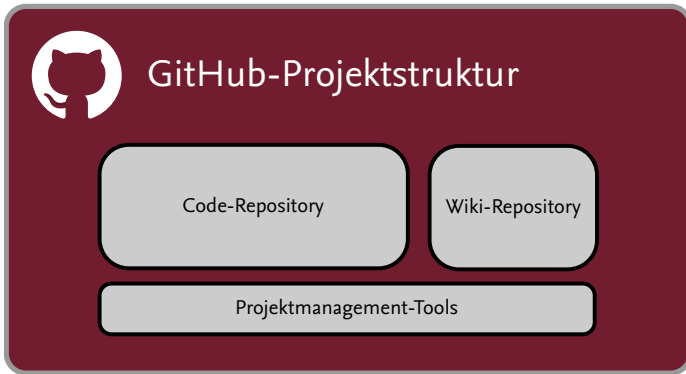
- Zentraler Server, auf dem alle Revisionen gespeichert werden
- Authentifizierung möglich (z. B. über Benutzername und Passwort)
- Ein Benutzer hat auf seinem Rechner eine Arbeitskopie
  - Synchronisation mit Server  $\Rightarrow$  Update der Arbeitskopie
  - Entwickler verändern ihre Arbeitskopie
  - Einchecken der Änderungen beim zentralen Server
- Spätestens bei einem Commit muss der Entwickler den aktuellen Stand vom Server beziehen



# Verteilte Versionskontrollsysteme: Funktion

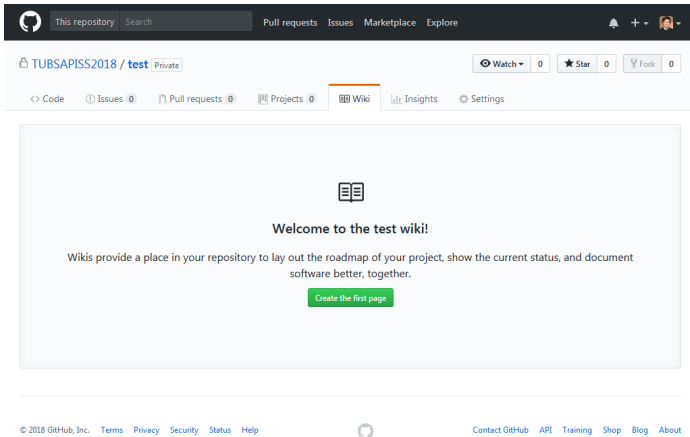
- Kein zentrales Repository erforderlich
  - Jeder Entwickler hat eine vollständige Kopie des Repositorys
  - Änderungen und Commits werden ins lokale Repository gepflegt
  - Synchronisation von Änderungen zwischen verschiedenen Entwicklern
- Ohne zentrales Repo ist der Quelltextaustausch umständlich
  - Hauptrepository ermöglicht einfachen Austausch des Quellcodes
- Kein Single-Point-of-Failure
  - Höhere Datensicherheit





# Projekt-Wiki initialisieren

Das Wiki kann primär online erstellt und bearbeitet werden.



# Situation

Ich möchte an einem existierendem Projekt weiterarbeiten  
oder das existierende Projekt verwenden.



# git clone

## Adresse

Die Adresse des Code-Repositorys auf GITHUB kann mit einem Klick auf die „Clone or Download“-Schaltfläche eingesehen und kopiert werden.

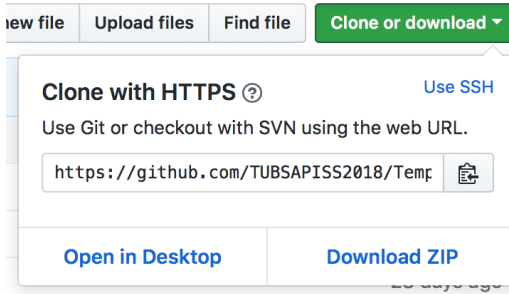


Abbildung 1: Adresse eines Code-Repositorys in GitHub

# git clone

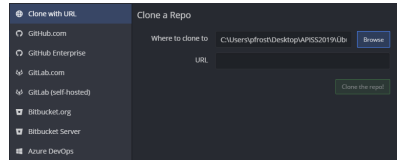
## SHELL

Über den Befehl **git clone** wird das gesamte Repository in ein neu erstelltes Verzeichnis geklont

### Listing 1: Repository klonen

```
git clone "https://  
Adresse.git"
```

## GITKRAKEN





# git clone

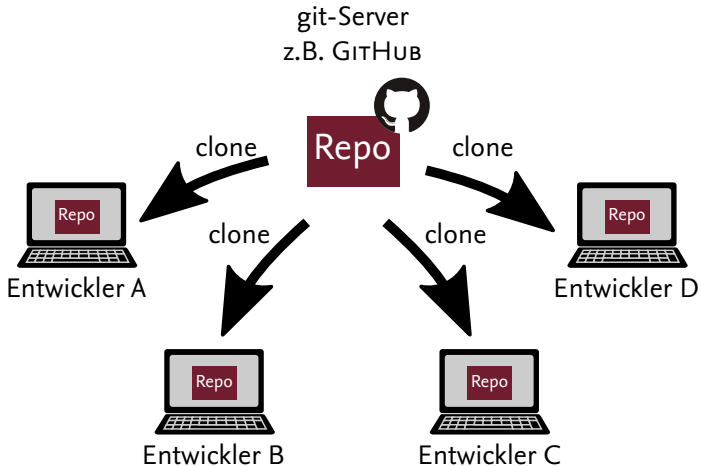


Abbildung 2: git clone eines GITHUB-Repositorys

# Situation

Ich möchte den aktuellen Stand einsehen und überprüfen, aber noch nicht in den aktuell aktiven Zweig überführen.



# git fetch

Über den Befehl **git fetch** wird der aktuelle Stand von dem Server heruntergeladen. Dabei bleibt das Arbeitsverzeichnis (engl. Working Directory) unverändert.

Listing 2: Repository synchronisieren

```
git fetch
```

Listing 3: In den Zweig wechseln

```
git checkout "Zweigname"
```

Ein anderer Zweig kann so überprüft und über **git merge** übernommen werden.

# Situation

Ich möchte den aktuellen Stand herunterladen und direkt in meinen aktiven Zweig überführen.



# git pull

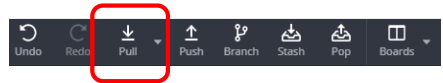
Der Befehl `git pull` fasst die Befehle `git fetch` und `git merge` zusammen.

## SHELL

Listing 4: Repository synchronisieren und Working Directory anpassen

```
git pull
```

## GITKRAKEN



## Wichtig

Änderungen müssen vor dem Ausführen von `git pull` über `git commit` gesichert werden

# Situation

Ich möchte meine Commits an den Server übertragen.



# git push

Über den Befehl **git push** können die Commits des aktiven lokalen Zweiges an den Server übertragen werden.

## SHELL

Listing 5: Commits hochladen

```
git push
```

## GITKRAKEN



**git push** kann nur dann angewendet werden, wenn das lokale Repository auf dem aktuellen Stand ist.

# git-Arbeitsfluss

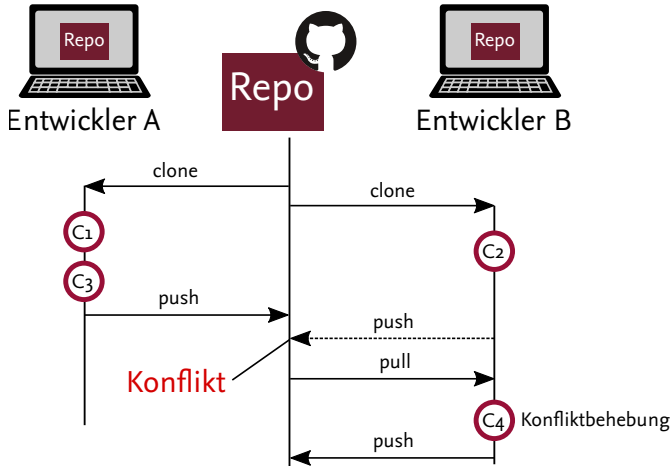


Abbildung 3: git-Arbeitsfluss



# Situation

Ich möchte die Revisionsnummer abfragen.



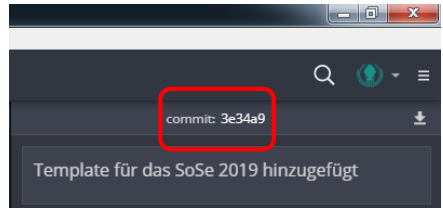
# git Revisionsnummer

## SHELL

Listing 6: Commit-Verlauf anzeigen

```
git log
```

## GITKRAKEN



## Wichtig

Die Revisionsnummer ist der Hashwert eines Commits

(z.B.: 3e34a9b5e217956238ccc7445f3114bab6dfe678)

# Aufgaben

1. Das Projekt-Wiki initialisieren
2. Das Projekt-Wiki klonen
3. Das Template-Wiki klonen
4. Inhalt des Template-Wikis lokal in das Projekt-Wiki kopieren
5. Das Template-Wiki lokal löschen
6. Den eigenen Namen in die Datei `Home.md` eintragen
7. Den lokalen Stand auf den Server übertragen
8. Falls erforderlich:
  - 8.1 Den Stand vom Server herunterladen
  - 8.2 Konflikte auflösen
  - 8.3 goto Aufgabe 7

Gibt es Fragen oder Anmerkungen zu dem Thema  
**Teamarbeit mit git?**



# Abgehakt

## Versionskontrolle mit GITHUB

Als Teilnehmer soll ich am Ende dieser Übung...

- ☒ Konflikte in git lösen können
- ☒ an bestehenden Projekten weiterarbeiten können
- ☒ das lokale Repository mit GITHUB synchronisieren können
- ☒ das Projekt-Wiki erstellen und mit Inhalt füllen können

# Aufgaben für die Nachbereitung

## Projekt-Code-Repo

1. Klont das Code-Repository.
2. Falls möglich: Mit der Programmierung beginnen.

## Projekt-Wiki-Repo

1. Wie können Bilder in das Projekt-Wiki hochgeladen werden?  
Siehe `Projektschema.md`
2. Fügt das Projektschema in das Projekt-Wiki hinzu.
3. Überträgt sämtliche Punkte für die Projektmappe in das Projekt-Wiki.

## Cheat Sheets

Cheat Sheets geben eine gute Übersicht über die verfügbaren git-Befehle.

- git und GitHub

[https://education.github.com/  
git-cheat-sheet-education.pdf](https://education.github.com/git-cheat-sheet-education.pdf)

- Dynamisches Cheat Sheet

<http://ndpsoftware.com/git-cheatsheet.html>

Vielen Dank für eure Aufmerksamkeit!