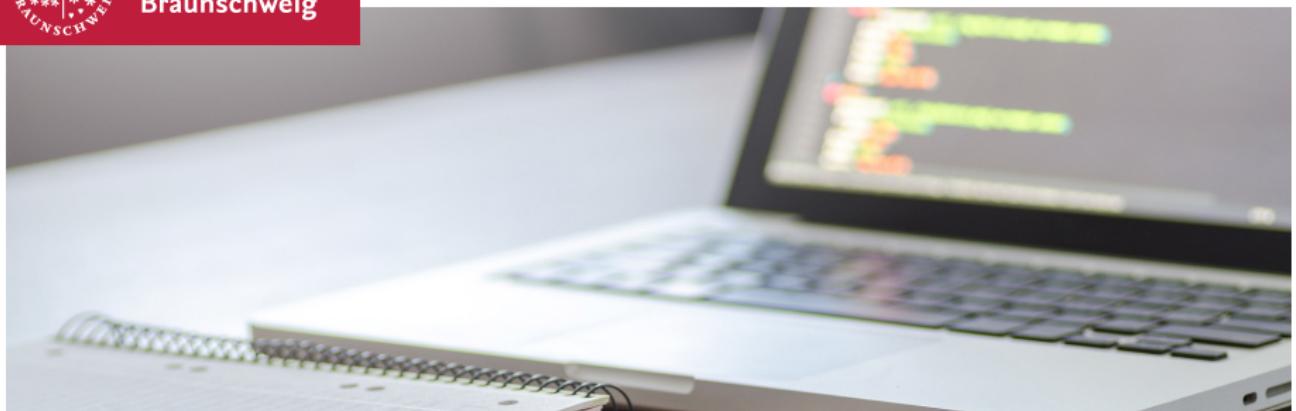




Technische  
Universität  
Braunschweig

Institut für  
Flugführung



# Entwicklungsumgebungen und Versionskontrolle

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,  
17. April 2018

# Agenda

- 03. April Einführung
- 10. April Softwareprojektmanagement
- 17. April Entwicklungstools**
- 24. April GitHub
- 08. Mai Einführung Arduino/Funduino
- 15. Mai Dateieingabe und -ausgabe
- 22. Mai **Exkursionswoche**
- 29. Mai Dokumentation und Bug-Reporting
- 05. Juni Einführung von Qt
- 12. Juni GUI-Erstellung mit Qt
- 19. Juni Anleitung erstellen
- 26. Juni **Projektarbeit**
- 03. Juli **Vorbereitung der Abgabe**
- 10. Juli Abgabe**



# Lehrziele

## Einführung API

Als Teilnehmer soll ich am Ende dieser Übung...

- wissen, was eine Toolchain ist
- den Nutzen einer IDE in der Softwareentwicklung kennen
- die bereitgestellte virtuelle Maschine nutzen können
- einzeln mit einer lokalen Versionsverwaltung arbeiten können



# Beispielprojekt

## Rahmenbedingungen

IFF: 55+ Mitarbeiter

250+ Kaffebezüge pro Woche

Zwei Preise: Kaffee, Kaffeespezialität

Abrechnung erfolgt per Strichliste

## Grobe Idee

Elektronisches Abrechnungssystem als  
Ersatz für die Strichliste.



# User-Stories und Zielkriterien

## User-Story

Als Mitarbeiter möchte ich meinen Kaffee automatisch über das Abrechnungssystem bezahlen können, damit dieser schneller den Kaffee beziehen kann.

## 4 Zielkriterien

1. Der Mitarbeiter wird anhand seiner Kaffeetasse identifiziert
2. Falls der Mitarbeiter genug Guthaben hat, werden ihm die Kaffeeoptionen angezeigt
3. Nachdem der Kaffee bezogen wurde, werden die Kosten für den Kaffee von dem Guthaben abgezogen
4. Der Restbetrag wird dem Mitarbeiter angezeigt



## IDEs



Die Toolchain

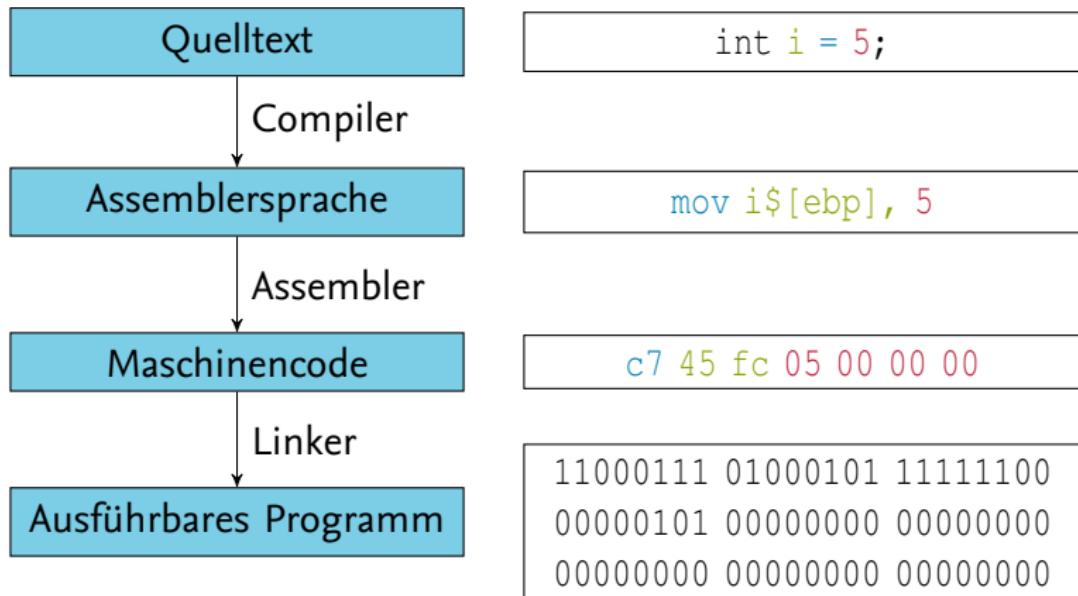
Entwicklungs-  
umgebungen

API-Tools

Projekt-  
erstellung



# Vom Quelltext zum ausführbaren Programm

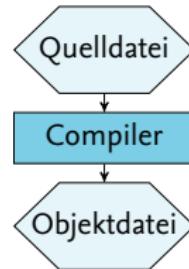


# Toolchain

## 1. KOMPILIEREN

Quelltext in Maschinencode  
umwandeln

→ Objektdatei

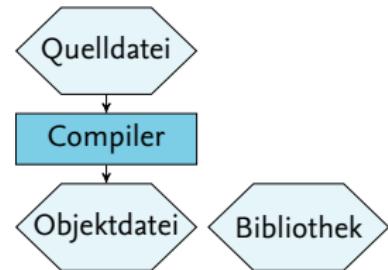


# Toolchain

## 1. KOMPILIEREN

Quelltext in Maschinencode  
umwandeln

→ Objektdatei



# Toolchain

## 1. KOMPILIEREN

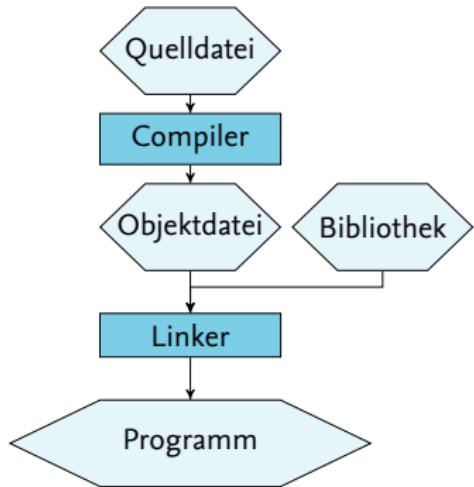
Quelltext in Maschinencode umwandeln

→ Objektdatei

## 2. LINKEN

Maschinencode und Bibliotheken zusammenfassen

→ Programm (.exe) oder Bibliothek (.lib)



# Toolchain

## 1. KOMPILIEREN

Quelltext in Maschinencode umwandeln

→ Objektdatei

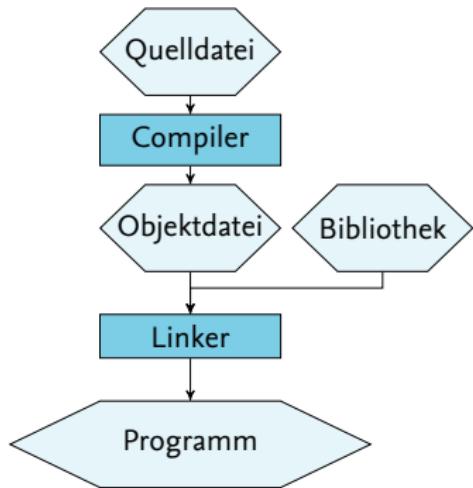
## 2. LINKEN

Maschinencode und Bibliotheken zusammenfassen

→ Programm (.exe) oder Bibliothek (.lib)

### ■ Bei Änderungen

- Quelltext kompilieren
- Erneutes Linken



# Erstellen einer ausführbaren Datei

- Quellcode erstellen:  
→ Quelldateien

main.cpp

eins.cpp/eins.h  
zwei.cpp/zwei.h

- Kompilierung:  

```
$ g++ -c eins.cpp zwei.cpp main.cpp
```

- Objektdateien

main.o  
eins.o  
zwei.o

- Linken:  

```
$ g++ -o Beispiel eins.o zwei.o main.o
```

- Programm

Beispiel.exe



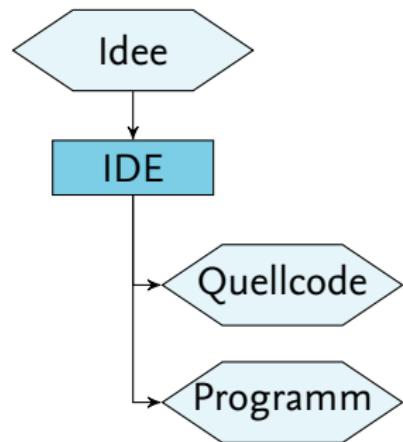
Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Toolchain?**



# Integrierte Entwicklungsumgebungen

## Integrated Development Environment (IDE)

- Gesamte Toolchain in einem Werkzeug
- Zur Programmierung optimierter Texteditor
  - Syntaxhighlighting & -überprüfung, Codevervollständigung
- Projektverwaltung
- Assistenten
  - Erstellung von Klassen, Benutzeroberflächen (GUI) etc.
- Integration weiterer Werkzeuge
  - z. B. Quellcodeverwaltung (Git)



# Integrierte Entwicklungsumgebungen II

Verschiedene IDEs sind jeweils für bestimmte Zielplatformen und Programmiersprachen optimiert.

	Visual Studio	Windows	.NET-Sprachen
	XCode	macOS	Entwicklung auf macOS
	Android Studio	Alle OS	Android-Apps
	IntelliJ	Alle OS	Java
	Qt Creator	Alle OS	C++ Entwicklung mit Qt
	CLion	Alle OS	C++



# Microsoft Visual Studio

## Vorteile:

- Insb. für MS Sprachen optimal  
→ .NET: Visual Basic, C#, F# ...
  - Erweiterbar für nahezu jede Programmiersprache  
→ C/C++ ...

## Nachteile:

- Hohe Komplexität
  - Sehr groß (ca. 30 GB)
  - Primär Windows als Zielplatform

Läuft unter: Windows  
Download über TU / MS Imagine

# Microsoft Visual Studio Code

## Multiplattform-Editor

## Vorteile:

- Sehr schlank
  - Mit Plug-Ins erweiterbar  
→ PLATFORMIO IDE für Arduino
  - GitHub-Erweiterung verfügbar

## Nachteile:

- Keine vollwrtige IDE  
→ Plug-Ins erforderlich
  - Compiler mssen z. T. selbst installiert werden.

Für: Windows, Linux, Mac  
[code.visualstudio.com](http://code.visualstudio.com)

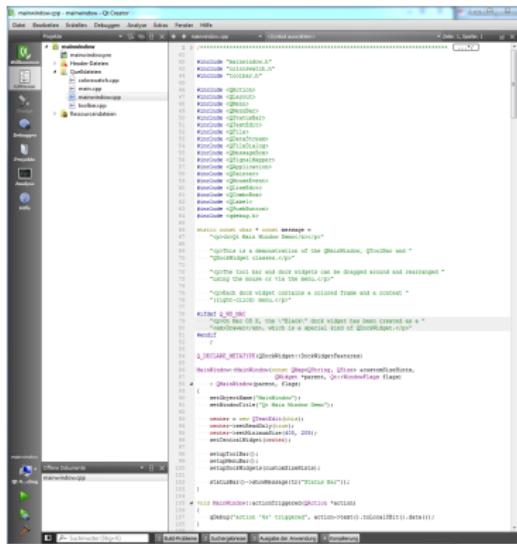
# Qt Creator



IDE für das Qt Framework

## Vorteile:

- Plattformübergreifende Programmierung
- Optimiert für Qt & C++
- Mehrere Compiler unterstützt
  - GCC (Linux)
  - MinGW (Windows)
  - Visual Studio Compiler



## Nachteile:

- Nicht einfach erweiterbar

Für: Windows, Linux, Mac  
[qt.io/download](http://qt.io/download) (Open Source)



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Entwicklungsumgebungen?**



# API-Entwicklungstools

Vorstellung *plattformübergreifender* Entwicklungstools.  
Nach Belieben kann aber auch jede andere IDE verwendet werden.

## Arduino-Entwicklung

VISUAL STUDIO CODE mit Plug-In PLATFORMIO IDE

Einfache Installation, IDE-Funktionalität

Ausführliche Vorstellung voraussichtlich am 8. Mai

## PC-Entwicklung mit C++ / Qt

QT CREATOR

Einfache Installation, auf Qt abgestimmt, integrierte Werkzeuge zur  
GUI-Erstellung



# Installationshinweise

# VS CODE & PLATFORMIO

Nach der Installation von VS Code muss das Plug-In PLATFORMIO IDE installiert werden.

1. In VS CODE auf die Schaltfläche für Erweiterungen klicken.
2. Nach PLATFORMIO IDE suchen.
3. Erweiterung installieren.



# Installationshinweise

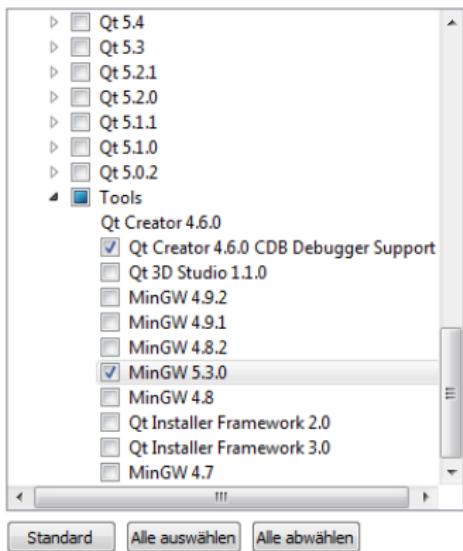
# Qt & Qt CREATOR

Während des Installationsprozesses müssen mindestens die folgenden Komponenten ausgewählt werden:

1. Qt > Qt 5.10.1 > MinGW 5.3.0 32 bit
2. Qt > Tools > Qt Creator 4.6.0
3. Qt > Tools > Qt Creator 4.6.0 CDB Debugger Support
4. Qt > Tools > MinGW 5.3.0

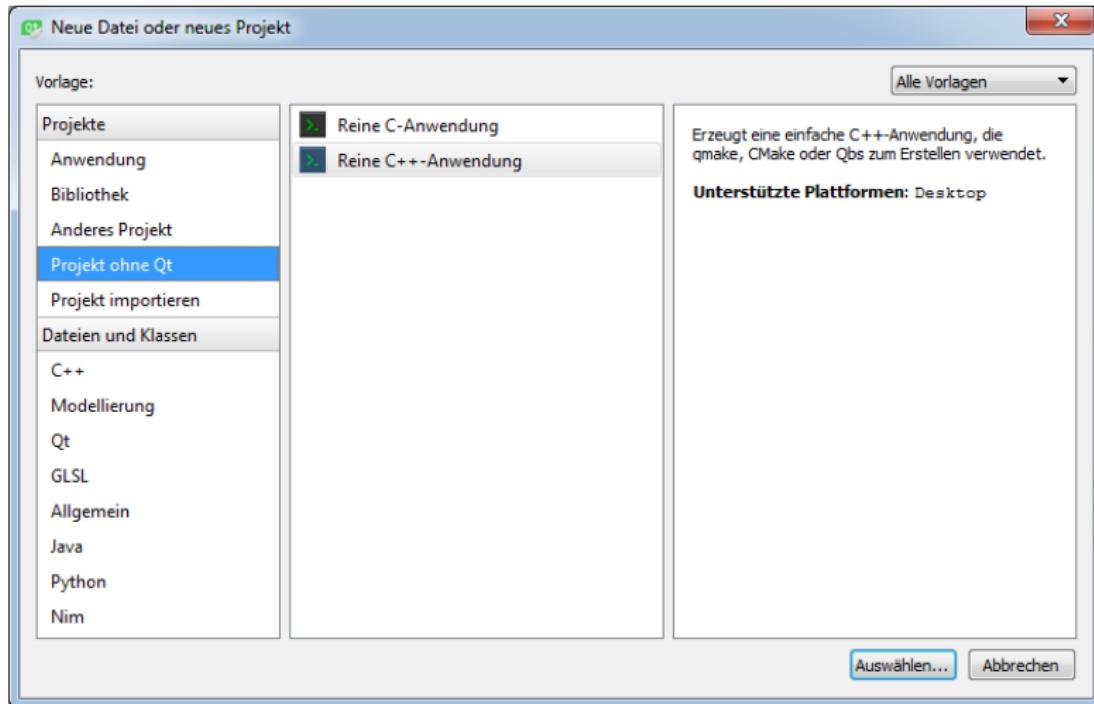
## Komponenten auswählen

Bitte wählen Sie die Komponenten aus, die Sie installieren möchten.



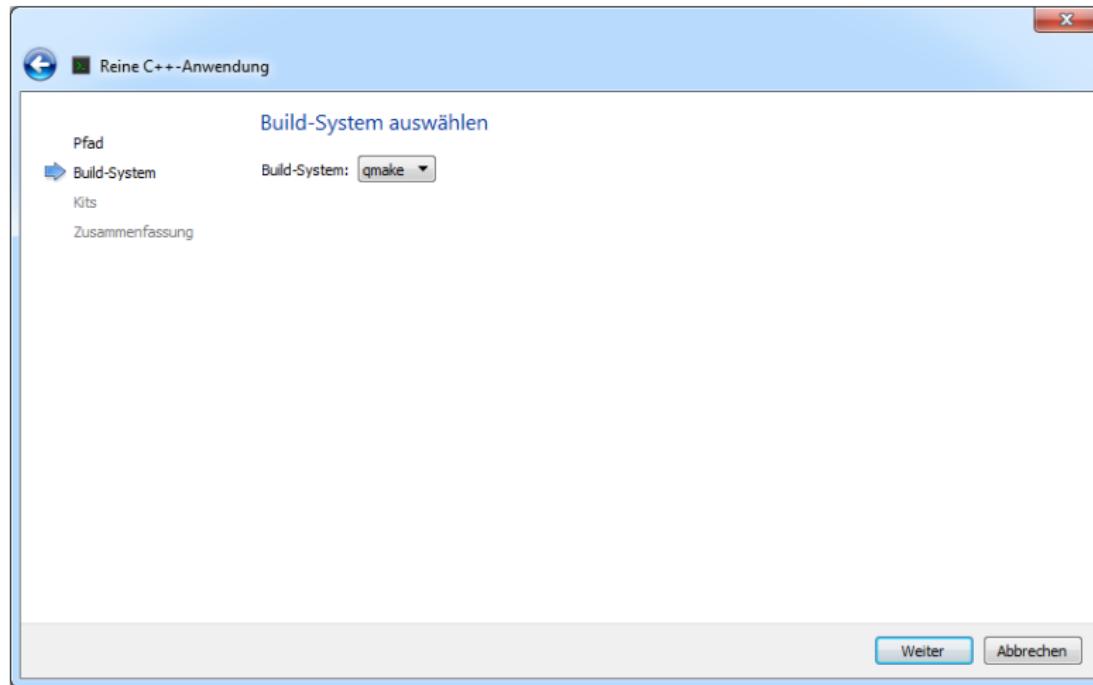
# Beispiel

# Erstes C++-Projekt



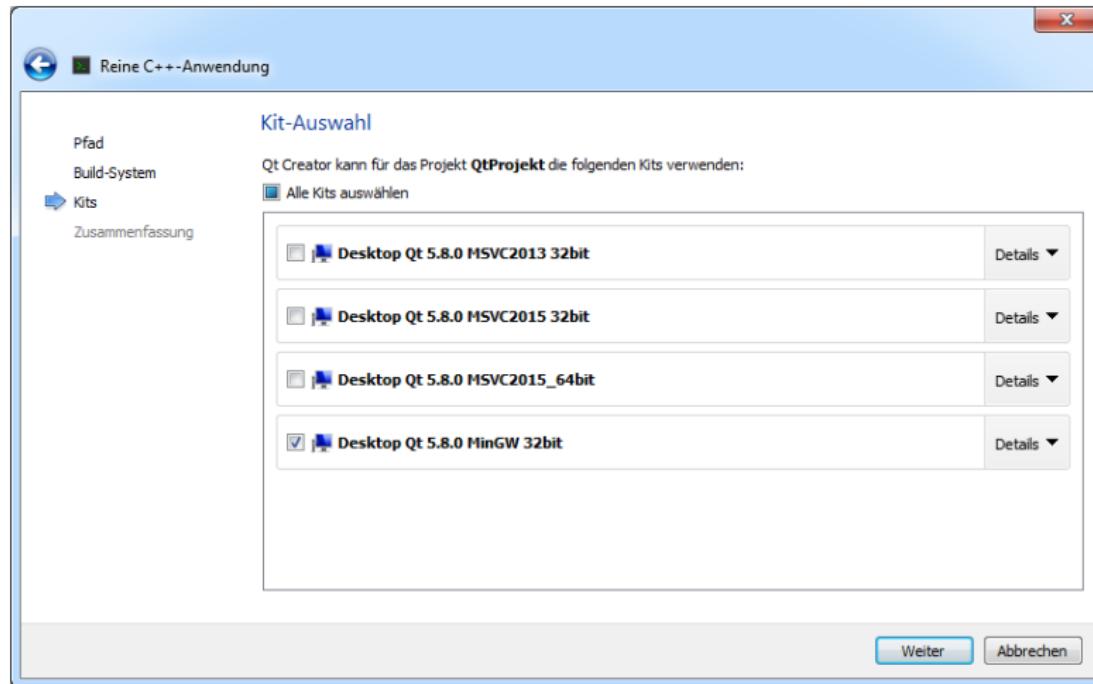
# Beispiel

# Erstes C++-Projekt



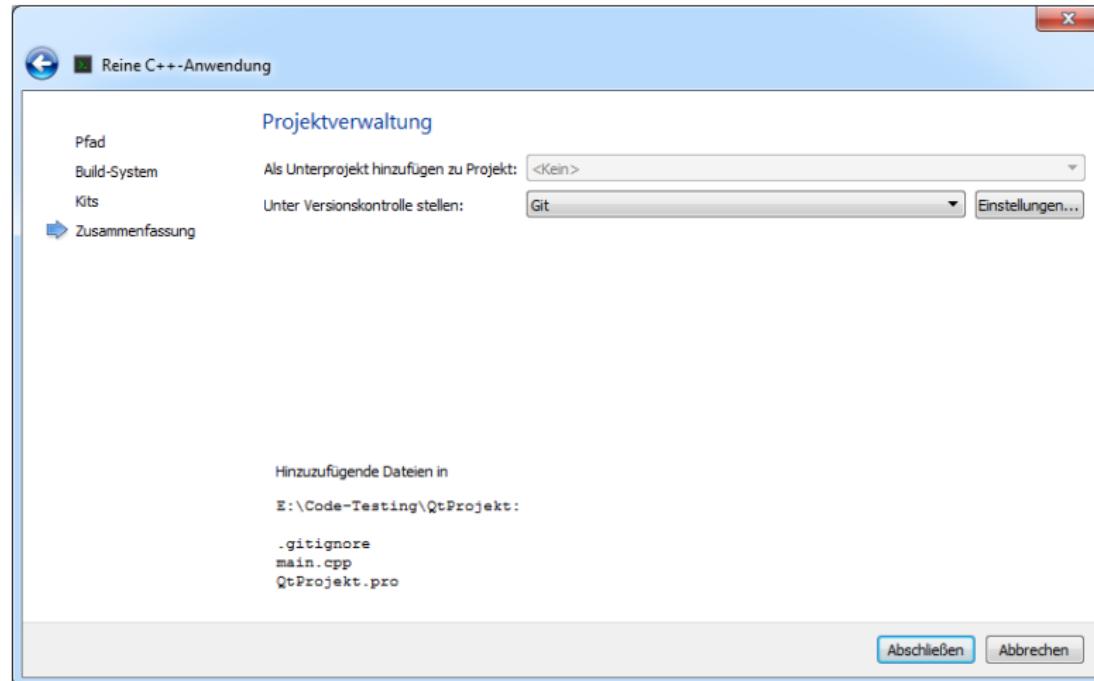
# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt

QtProjekt.pro [master] - API - Qt Creator

Datei Bearbeiten Erstellen Debuggen Analyse Extras Fenster Hilfe

Projekte Willkommen Editieren Design Debug Projekte Hilfe

QtProjekt

Suchmuster (Strg+K)

1 TEMPLATE = app  
2 CONFIG += console c++11  
3 CONFIG -= app\_bundle  
4 CONFIG -= qt  
5  
6 SOURCES += main.cpp  
7 |



# Beispiel

# Erstes C++-Projekt

The screenshot shows the Qt Creator IDE interface. The title bar reads "main.cpp [master] - API - Qt Creator". The menu bar includes Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, and Hilfe. The left sidebar has sections for Willkommen, Editieren, Design, Debug, Projekte, and Hilfe, with "QtProjekt" selected. The central Projects view shows "QtProjekt [master]" containing "QtProjekt.pro" and "Quelldateien/main.cpp". The main editor area displays the following code:

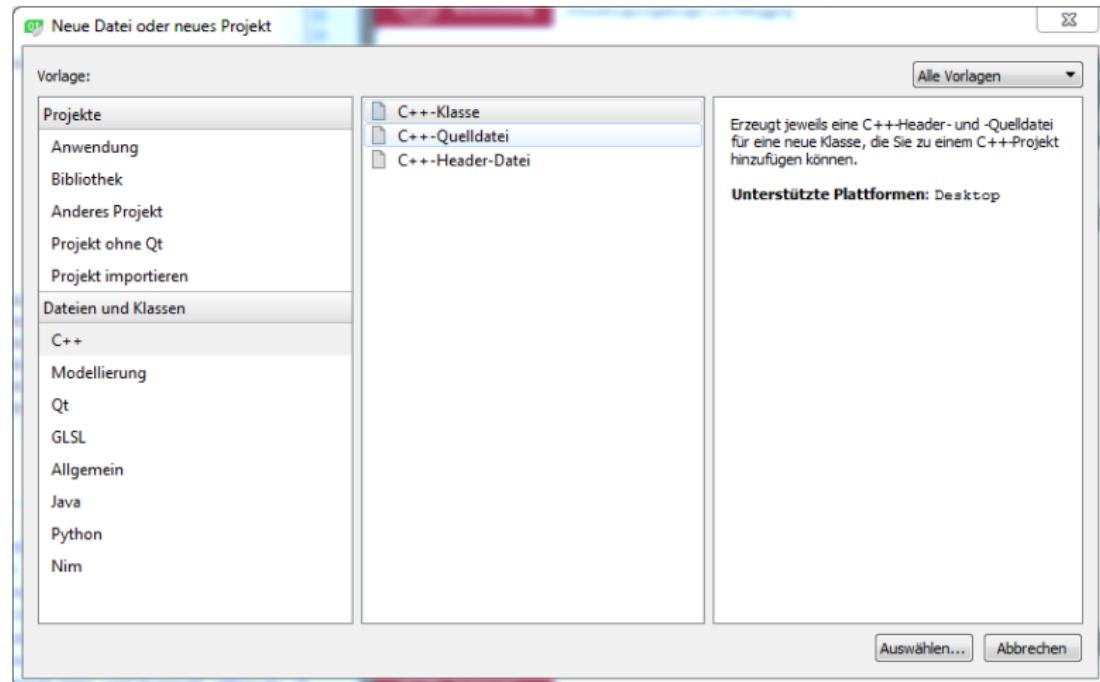
```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(int argc, char *argv[])
6 {
7     cout << "Hello World!" << endl;
8     return 0;
9 }
10
```

At the bottom, there is a search bar "Suchmuster (Strg+K)" and several tool buttons: Build-P..., Sucher..., Ausga..., Kompili..., Debug... .



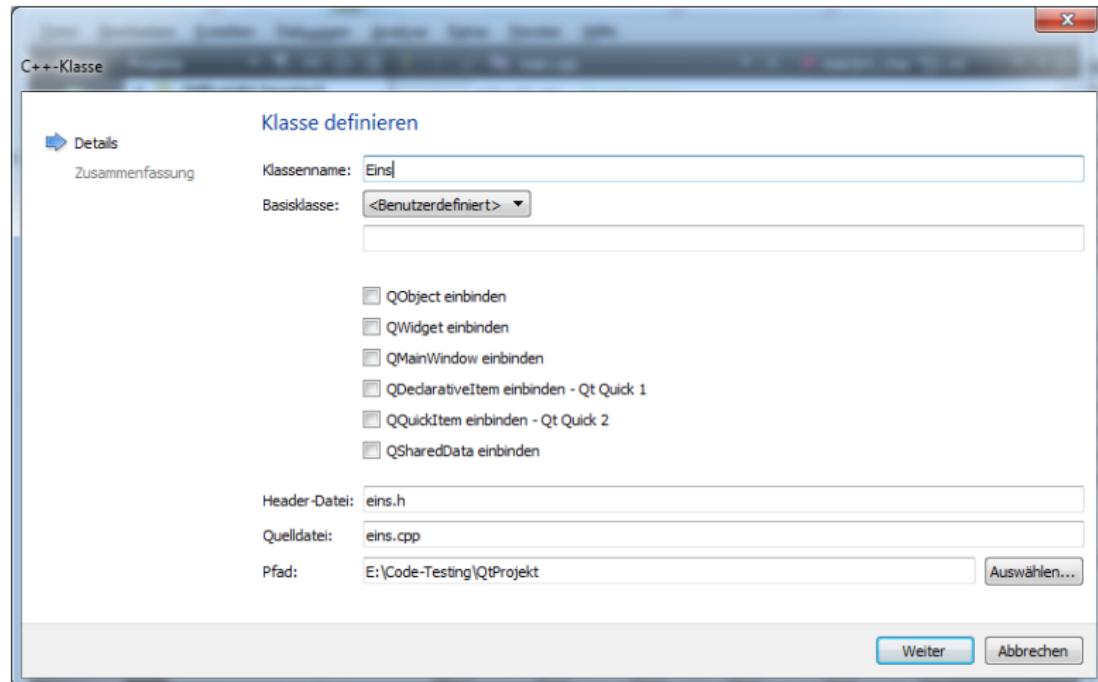
# Beispiel

# Erstes C++-Projekt



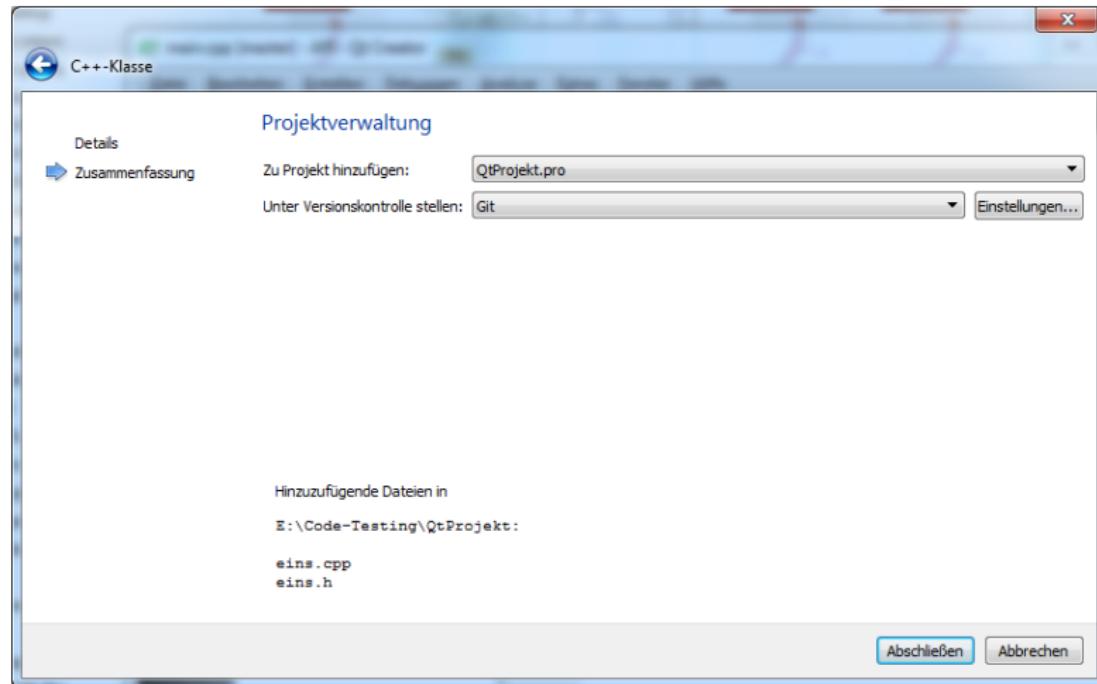
# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt

The screenshot shows the Qt Creator interface with the title bar "QtProjekt.pro [master] - API - Qt Creator". The menu bar includes Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, and Hilfe. The toolbar on the left has icons for Willkommen, Editieren, Design, Debug, Projekte, and Hilfe. The central area displays a project tree under "QtProjekt [master]" with "Header-Dateien" containing "eins.h" and "Quelldateien" containing "eins.cpp" and "main.cpp". Below the tree is a code editor showing the following .pro file content:

```
1 TEMPLATE = app
2 CONFIG += console c++11
3 CONFIG -= app_bundle
4 CONFIG -= qt
5
6 SOURCES += main.cpp \
7     eins.cpp
8
9 HEADERS += \
10     eins.h
11
```

The bottom of the window features a search bar "Suchmuster (Strg+I)" and several tabs labeled Build-P..., Sucher..., Ausga..., Kompili..., and Debug...".



# Beispiel

# Erstes C++-Projekt

The screenshot shows the Qt Creator interface with the following details:

- Title Bar:** eins.h [master] - API - Qt Creator
- Menu Bar:** Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, Hilfe
- Toolbars:** Willkommen, Editieren, Design, Debug, Projekte, Hilfe, QtProjekt, Debug, Run, Stop, Build.
- Project Explorer:** QtProjekt [master] (QtProjekt.pro), Header-Dateien (eins.h), Quelldateien (eins.cpp, main.cpp).
- Code Editor:** eins.h content:

```
1 #ifndef EINS_H
2 #define EINS_H
3
4
5 class Eins
6 {
7 public:
8     Eins();
9 };
10
11#endif // EINS_H
```
- Bottom Bar:** Suchmuster (Strg+K), Build-P..., Sucher..., Ausga..., Kompili..., Debug...



Gibt es Fragen oder Anmerkungen zu dem Thema  
IDEs?



# Abgehakt

## Einführung API

Als Teilnehmer soll ich am Ende dieser Übung...

- wissen, was eine Toolchain ist
- den Nutzen einer IDE in der Softwareentwicklung kennen
- die bereitgestellte virtuelle Maschine nutzen können
- einzeln mit einer lokalen Versionsverwaltung arbeiten können



VirtualBox



Virtuelle  
Maschine?

Einrichtung



# Virtuelle Maschine (VM)

Eine virtuelle Maschine emuliert einen Rechner, so dass auf dieser Maschine ein Betriebssystem installiert und ausgeführt werden kann.

## Vorteile

- Sandkastenumgebung
- Erstellung von Applikationen für andere Plattformen möglich
- Übertragbare Entwicklungsumgebung

## Nachteile

- Ressourcen müssen geteilt und fest zugewiesen werden
- Geringere Leistungsfähigkeit



# Einrichtung der API-VM

Für die Verwendung der API-VM sind die folgenden Schritte erforderlich:

1. Installation von VirtualBox

<https://www.virtualbox.org/wiki/Downloads>

2. Herunterladen der VM (siehe E-Mail der Anmeldebestätigung)
3. VM hinzufügen (Maschine → Hinzufügen)
4. Geräteeinstellungen anpassen:

Name der Maschine Ubuntu

Arbeitsspeicher minimum 2 GB

Festplatte Bereits existierende Festplatte,

Pfad zur heruntergeladenen VDI-Datei



Gibt es Fragen oder Anmerkungen zu dem Thema  
**VirtualBox?**



# Abgehakt

## Einführung API

Als Teilnehmer soll ich am Ende dieser Übung...

- wissen, was eine Toolchain ist
- den Nutzen einer IDE in der Softwareentwicklung kennen
- die bereitgestellte virtuelle Maschine nutzen können
- einzeln mit einer lokalen Versionsverwaltung arbeiten können



git



Motivation

git

Übung



# Versionskontrolle?

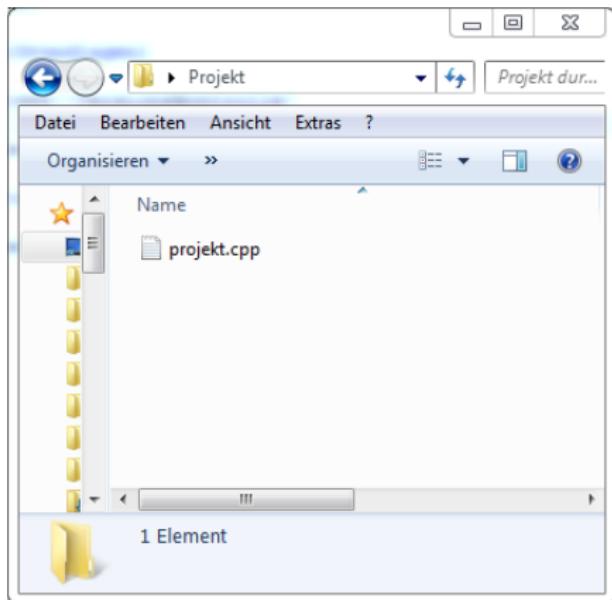


Abbildung 1: Versionsverwaltung durch Copy und Paste



# Versionskontrolle?

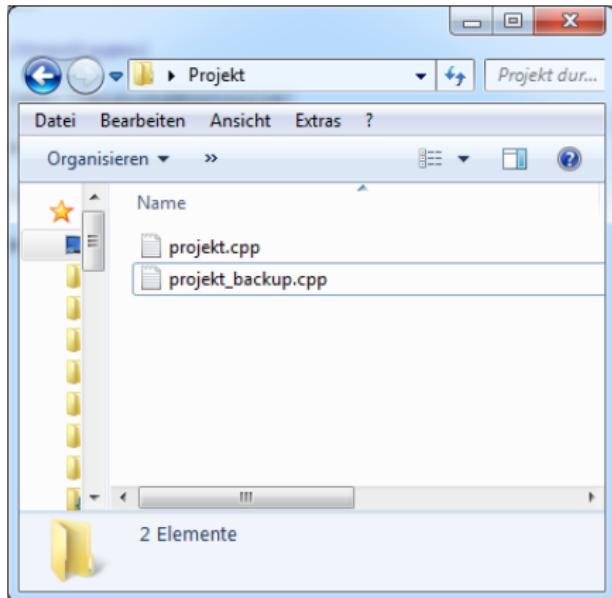


Abbildung 1: Versionsverwaltung durch Copy und Paste

# Versionskontrolle?

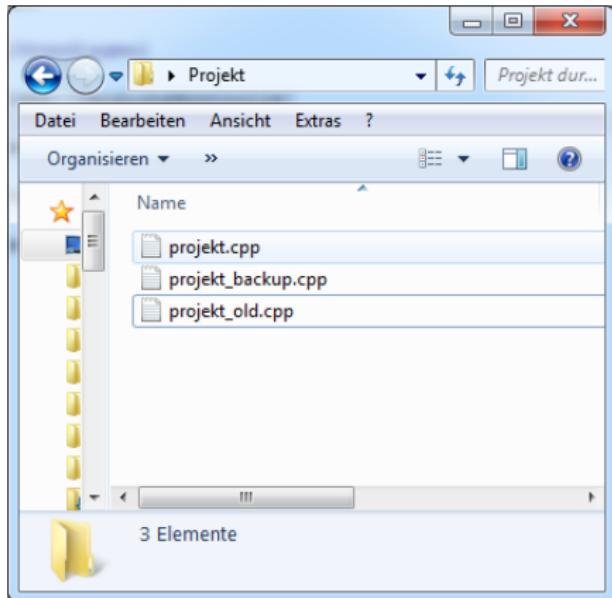


Abbildung 1: Versionsverwaltung durch Copy und Paste



# Versionskontrolle?

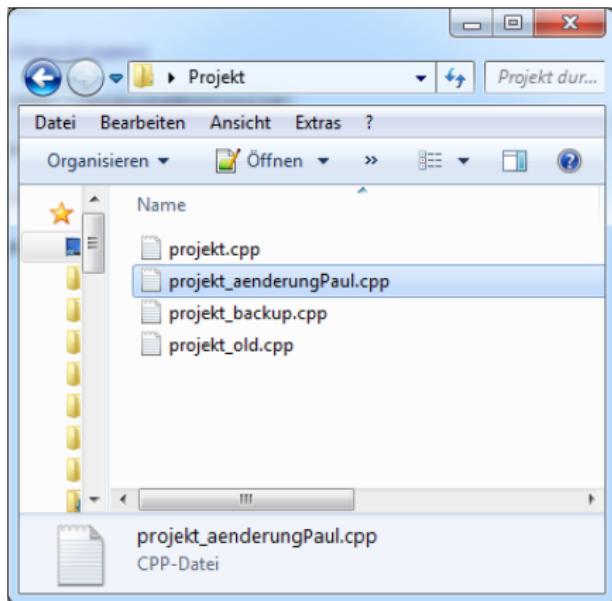


Abbildung 1: Versionsverwaltung durch Copy und Paste

# Versionskontrolle?

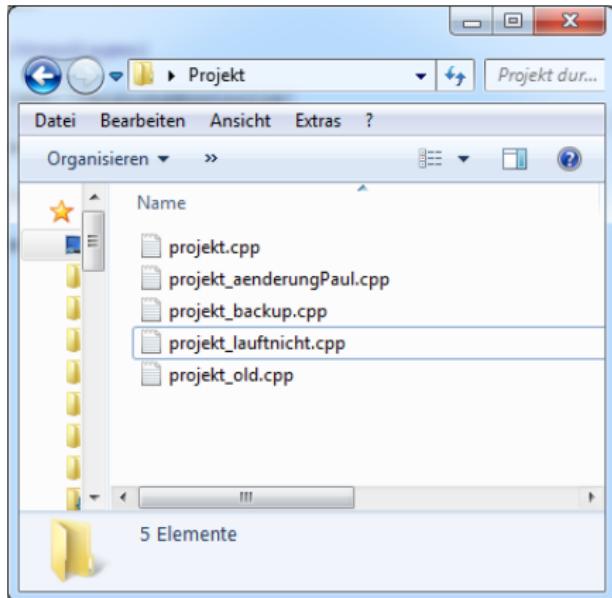


Abbildung 1: Versionsverwaltung durch Copy und Paste



# Versionskontrolle?

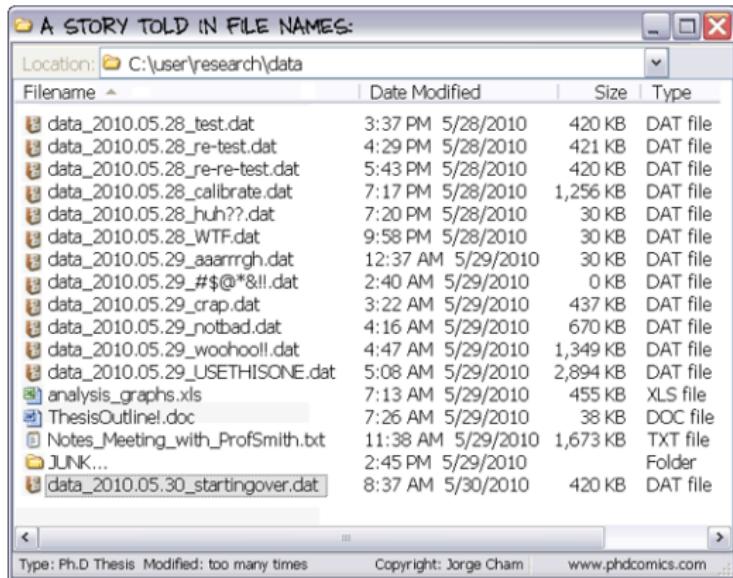


Abbildung 2: Versionierung mit Dateinamen  
“Piled Higher and Deeper” by Jorge Cham  
<http://www.phdcomics.com/comics.php?f=1323>

# Überblick über den Quelltext

2.3.2      2.1?  
1.1.4  
2.0    1.3  
              3.x?  
              3.2

- Unstrukturierte Softwareentwicklung
  - Übersicht über Quelltext fehlt
  - Typische Fragen
    - „Hat noch jemand von der Datei eine alte Version? Ich habe was geändert, und das klappt nicht!“
    - „Hattest du da gerade auch was geändert? Das habe ich jetzt überschrieben!“
    - „Wer hat denn den Code geschrieben?!“
- ⇒ Ordnender Prozess für die Bearbeitung von Quelltext nötig



# Ziele geordneter Entwicklung

- Parallelle Bearbeitung von Quelltext
  - Verschiedene Entwickler
  - Unterschiedliche Computer/Arbeitsplätze
  - Verschiedene Betriebssysteme
- Nachvollziehbarkeit von Änderungen
  - Dokumentation des Bearbeiters
  - Ausschluss von Änderungen
  - Zurücksetzen auf frühere Dateiversion
- Releaseverwaltung (Veröffentlichung)
  - Markierung von Versionsständen
  - Archivierung von Veröffentlichung

⇒ Versionsverwaltung notwendig

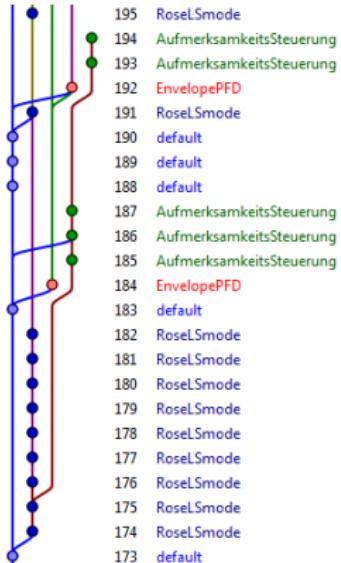


Abbildung 3: Versionsbaum

# git?



git ist ein **verteiltes Versionsverwaltungsprogramm**, das Entwicklern Folgendes ermöglicht:

- Schnelles Sichern von Dateien
- Paralleles Arbeiten am Quelltext
- Verwaltung von Versionsständen
- Einsehbarkeit und Wiederherstellbarkeit aller Versionsstände
- Integritätsprüfung
- Austausch von Quellcode



# Verwendung von git über die Kommandozeile

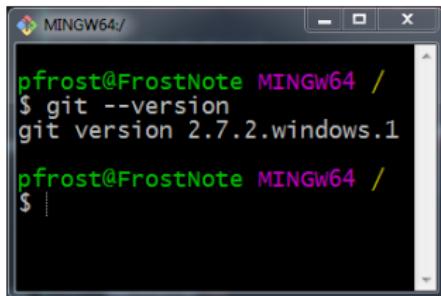
git ist ein Kommandozeilenprogramm und kann über Argumente gesteuert werden.

## Vorteile:

- Benötigt den geringsten Speicherplatz
- Nutzung über IDEs möglich
- Weite Verfügbarkeit

## Nachteile:

- Bedienung ist nicht intuitiv



```
p@frost@FrostNote MINGW64 /  
$ git --version  
git version 2.7.2.windows.1  
  
p@frost@FrostNote MINGW64 /  
$
```

Link:

<https://git-scm.com/downloads>



# Verwendung von git über GitHub Desktop

GITHUB DESKTOP liefert eine graphische Oberfläche für die Verwendung von git.

## Vorteile:

- git wird automatisch mitinstalliert
- Einfache Bedienung

## Nachteile:

- git steht erstmal nur GITHUB DESKTOP zur Verfügung
- Wenige git-Befehle verfügbar



Link:

<https://desktop.github.com/>

# Verwendung von git über weitere Programme

Auf der folgenden Seite sind weitere Programme für die Verwendung von git aufgelistet:

Link: <https://git-scm.com/downloads/guis>



# Lösungen

Falls Teile der Aufgaben nicht gelöst werden können, kann die folgende Datei heruntergeladen und extrahiert werden.

<https://github.com/TUBSAPISS2018/API-Materialien/Beispiele/Repo/beispielRepo.zip>

Die Lösungen der einzelnen Aufgaben befinden sich in den entsprechenden Unterordnern.



# Aufgaben

1. git-Repository erstellen
2. Textdatei-A.txt erstellen und in dem Repository sichern
3. Einen Zweig „dev“ anlegen
4. Textdatei-B.txt erstellen
5. Textdatei-B.txt in dem dev-Zweig sichern
6. Den Text in Textdatei-A.txt ändern und in dem dev-Zweig sichern
7. In den master-Zweig wechseln
8. Was ist mit den Textdateien passiert?
9. Den dev-Zweig in den master-Zweig überführen



# Situation

Ich habe noch nie mit einem Repository gearbeitet...





# git config

Vor der Arbeit mit einem Repository müssen die Benutzerdaten eingestellt werden.

## Listing 1: Entwicklernamen einrichten

```
git config --global user.name "Vor- Nachname"
```

## Listing 2: Entwickler-E-Mail einrichten

```
git config --global user.email "v.n@tu-bs.de"
```

Die E-Mail-Adresse sollte identisch mit für GitHub verwendeten E-Mail-Adresse sein.





# git config

Vor der Arbeit mit einem Repository müssen die Benutzerdaten eingestellt werden.

## Configure Git

This is used to identify the commits you create. Anyone will be able to see this information if you publish commits.

Name

Paul Frost

Email

p.frost@tu-braunschweig.de



Continue

Cancel

Abbildung 4: GitHub Desktop Konfiguration

Die E-Mail-Adresse sollte identisch mit für GitHub verwendeten E-Mail-Adresse sein.



# Situation

Ich möchte ein Repository erstellen, weil für die Übungsaufgaben noch kein Repository existiert...



# git init



## Das erste Repository erstellen

Listing 5: Repository erstellen

```
git init "NameDesRepositories"
```

Listing 6: In den erstellten Ordner wechseln

```
cd "NameDesRepositories"
```



# git init



Das erste Repository erstellen

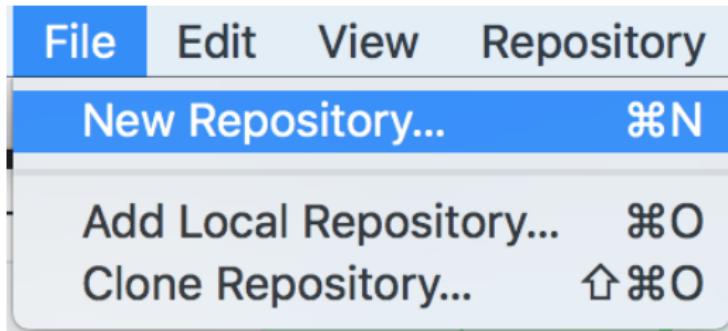


Abbildung 5: GitHub DESKTOP Repository erstellen



# git init

## Das erste Repository erstellen

Create a New Repository X

Name

Description

Local Path  
 Choose...

Initialize this repository with a README

Git Ignore  
 Choose...

License  
 Choose...

Cancel Create Repository

Abbildung 5: GitHub DESKTOP Repository erstellen



# .git

Mit dem Befehl `git init` wird ein versteckter `.git`-Ordner angelegt

- Der `.git`-Ordner ist das eigentliche Repository
- `git` speichert sämtliche Revisionen in diesem Ordner
- Das Repository kann nur wachsen

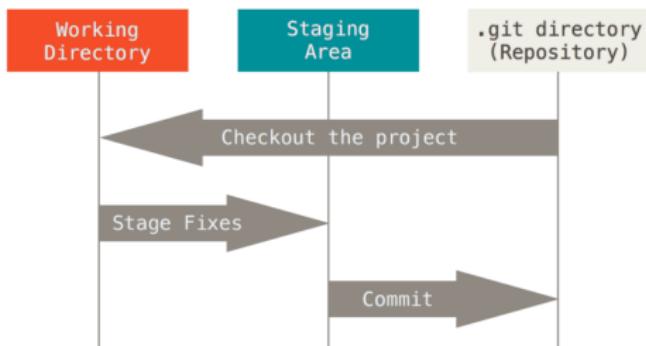


Abbildung 6: Arbeitsverzeichnis, Staging-Area und Git Verzeichnis [1]

Der `.git`-Ordner darf nicht gelöscht werden, wenn das lokale Repository weiter zur Verfügung stehen soll



# git Arbeitsfluss

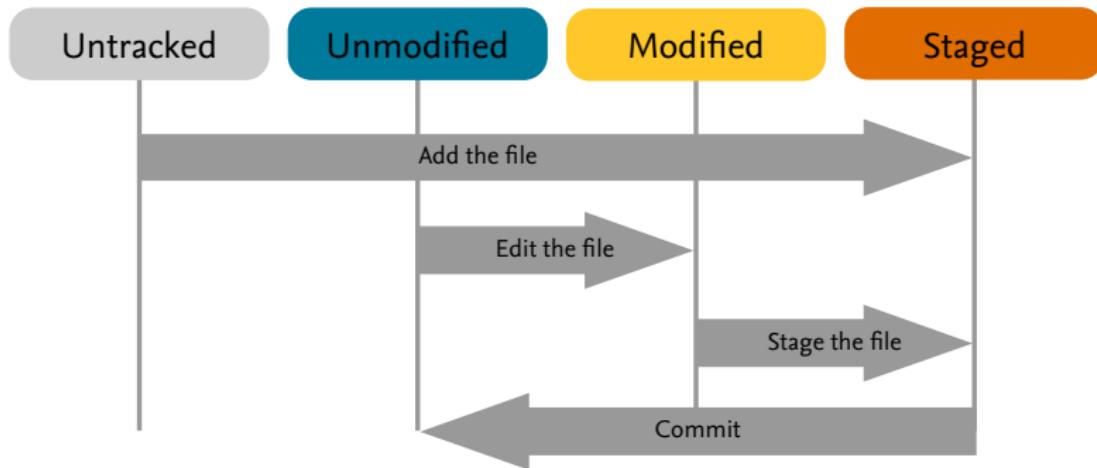
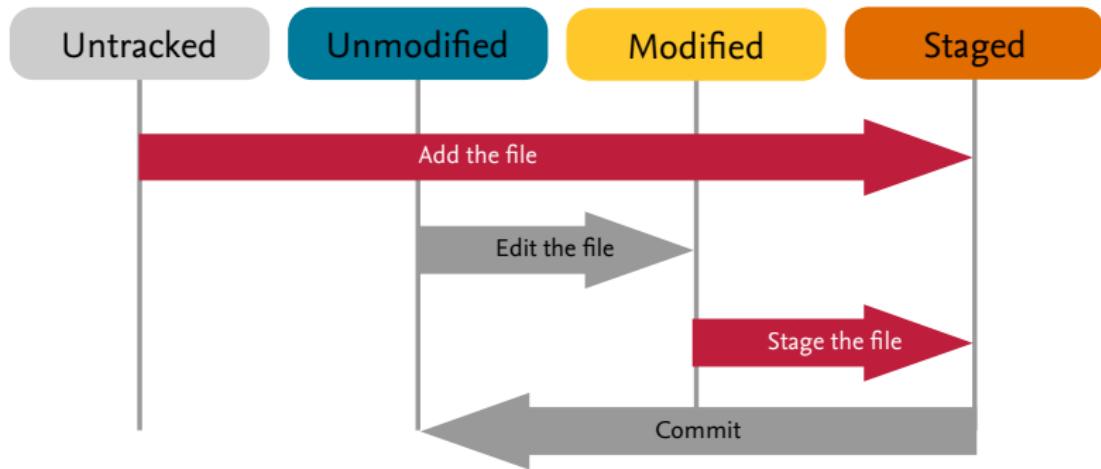


Abbildung 7: git workflow [1]



# git Arbeitsfluss

Dateien oder Änderungen, die im Repository gesichert werden sollen, müssen zunächst zur Staged-Ebene hinzugefügt werden.



# Situation

Ich habe die ersten Dateien in dem Repository-Ordner erstellt oder hinzukopiert und möchte diese versionieren bzw. sichern...





# git add

Über den Befehl **git add** können Dateien und Änderungen in die Staged-Ebene hinzugefügt werden.

Listing 11: Datei in die Staged-Ebene hinzufügen

```
git add "Dateiname.h"
```

Es können mehrere Dateien gleichzeitig hinzugefügt werden über:

**git add** -all

Alle Dateien aus der Untracked-Ebene und Modified-Ebene

**git add** \*

Alle Dateien und Unterordner des aktuell aktiven Ordners

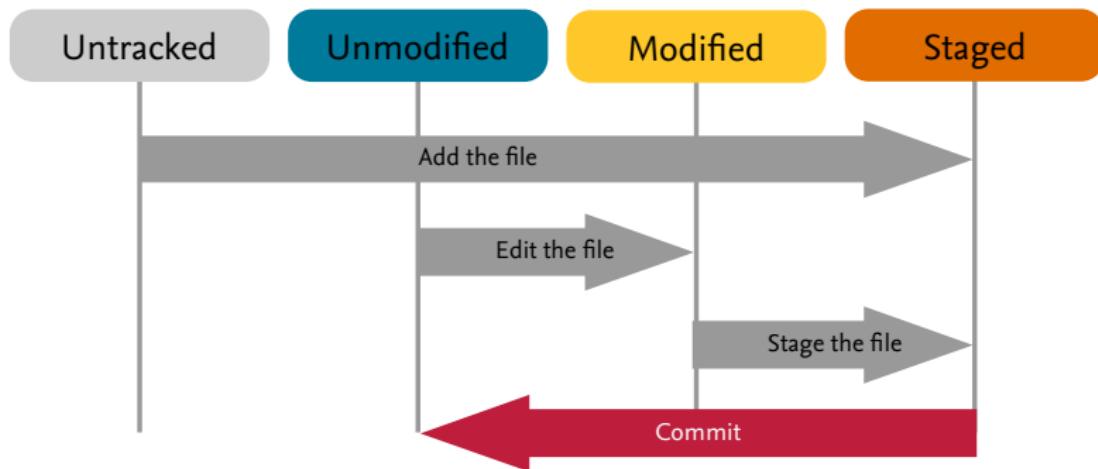
**git add** Dateiliste

Liste von Dateien getrennt durch ein Leerzeichen



# git Arbeitsfluss

Mit einem Commit wird alles aus der Staged-Ebene in das Repository überführt und als Revision gesichert.





# git commit

Der Befehl **git commit** sichert den aktuellen Stand in den aktiven Zweig

Listing 12: Daten

```
git commit -m "Beschreibung des Commits"
```

Die Option **-m** setzt mit der nachfolgenden Zeichenkette die Commit-Nachricht.

Die Commit-Nachricht ist bewertungsrelevant!





# git commit

Der Befehl **git commit** sichert den aktuellen Stand in den aktiven Zweig

The screenshot shows a Git commit interface. At the top, it displays the current repository as "beispielRepo" and the current branch as "master". Below this, there are two tabs: "Changes" (which is selected) and "History". Under the "Changes" tab, there is a list of changes: "1 changed file" and "TextdateiA.txt". A green "+" button is located next to the file name. In the bottom right corner of the changes list, there is a small blue box containing the number "1". Below the changes list, there is a "Summary" section with a placeholder for a user profile picture and a text input field labeled "Summary". Below that is a "Description" section with a large empty text area. At the bottom of the interface is a blue button labeled "Commit to master".

Die Commit-Nachricht ist bewertungsrelevant!



# Situation

Ich möchte risikofrei an einem Bug oder neuen Feature arbeiten...



# Zweige

Eine elementare Funktion von git ist die parallele Entwicklung mithilfe von Zweigen.

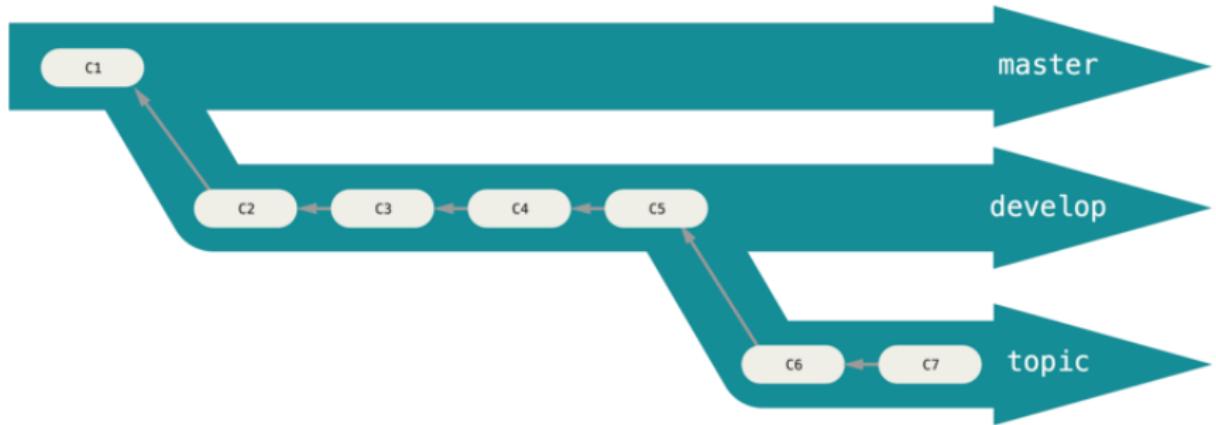


Abbildung 8: Zweige in git [1]





# git checkout

Listing 14: Neuen Zweig erstellen und in diesen Zweig wechseln

```
git checkout -b "Name_neuer_Zweig"
```

Listing 15: Zu einem existierenden Zweig wechseln

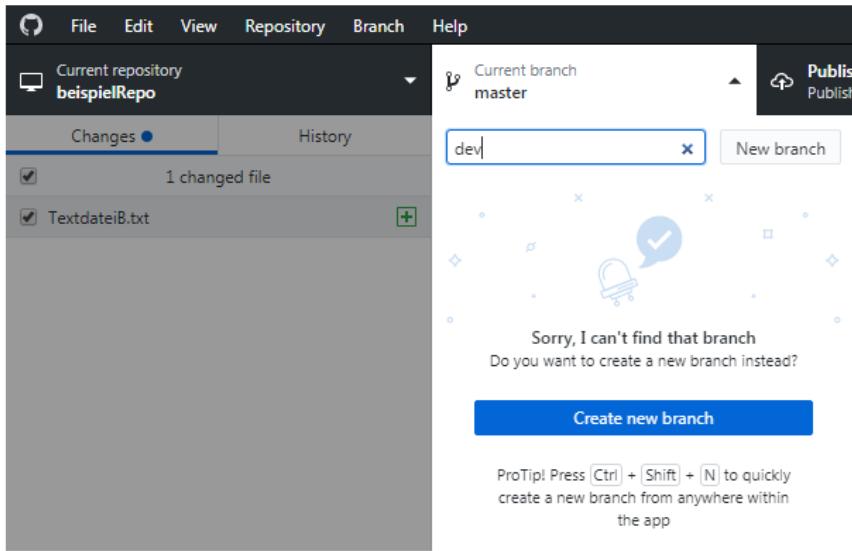
```
git checkout "Name_bestehender_Zweig"
```

Alle Änderungen an bereits versionierten Dateien müssen vorher mit einem Commit gesichert werden.





# git checkout



Alle Änderungen an bereits versionierten Dateien müssen vorher mit einem Commit gesichert werden.



# Situation

Ich habe mich vertan...





# Unversionierte Änderungen rückgängig machen

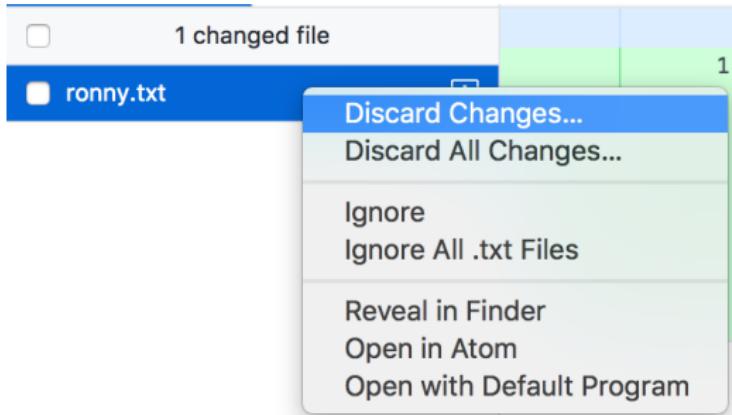
Der Befehl **git checkout** kann gleichzeitig dafür verwendet werden, Änderungen bis zu dem letzten Commit wiederherzustellen. Hierzu wird die Option **--** als Option vor einer Liste mit Dateien gesetzt.

Listing 18: Unversionierte Änderungen rückgängig machen

```
git checkout -- <Dateien>
```



# Unversionierte Änderungen rückgängig machen



# Situation

Die Arbeit an dem Feature oder Bug ist abgeschlossen und soll übernommen werden...



# Zusammenführen von Inhalten

Sobald ein Zweig getestet wurde und seinen Zweck erfüllt, sollte dieser in den Hauptzweig überführt werden.

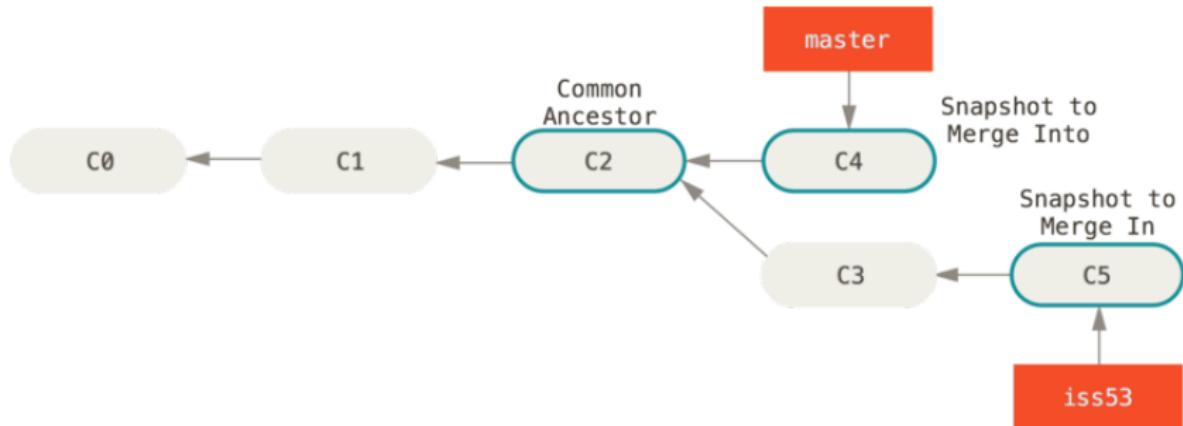


Abbildung 9: Vor einem Merge [1]



# Zusammenführen von Inhalten

Sobald ein Zweig getestet wurde und seinen Zweck erfüllt, sollte dieser in den Hauptzweig überführt werden.

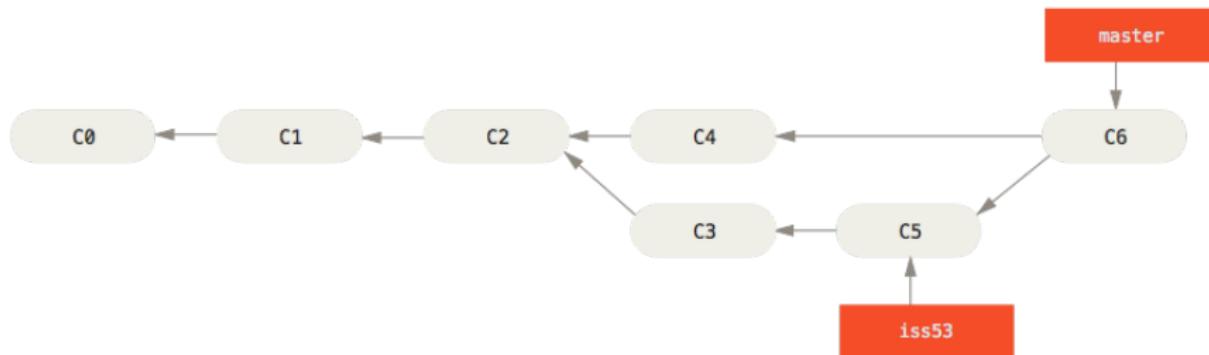


Abbildung 9: Nach einem Merge [1]

# git merge



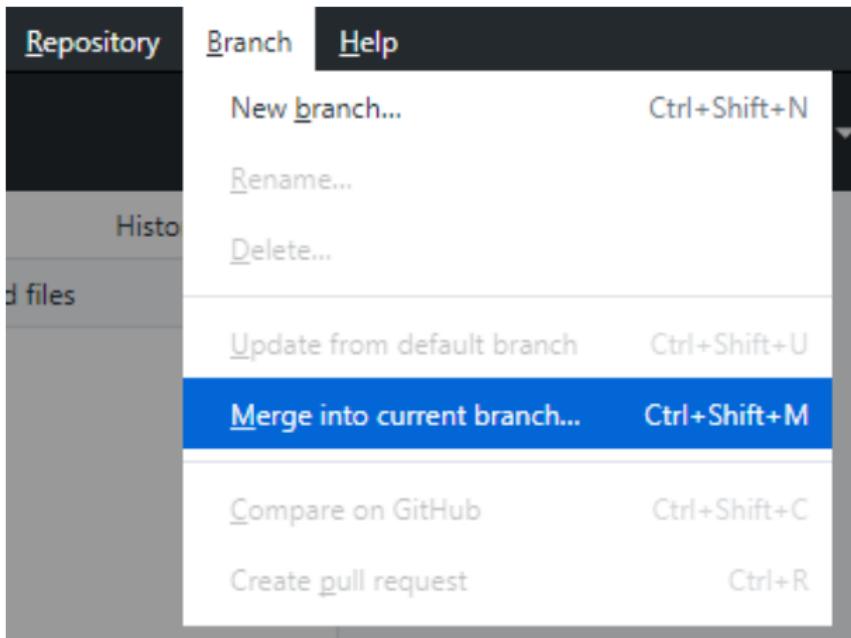
Listing 20: Zusammenführen von Zweigen

```
git merge "Name_Zweig"
```

In diesem Fall werden die Änderungen aus „Name\_Zweig“ in den aktuell aktiven Zweig überführt.



# git merge



# Ignore-Datei

# Dateien ignorieren

Listing 22: Ausschluss von Dateien über die Datei .gitignore

```
ordner/
datei.tex
*.exe
Dateien_mit*
```

Unter Umständen sinnvoll für:

- Binärdateien
- Temporäre Dateien (\*.log)
- Gebaute Dateien (\*.dll, \*.exe)

Nur neue Dateien werden ignoriert.



# Weitere wichtige git-Befehle

- git status** Hilfmittel bei der Überprüfung der Dateizustände
- git log** Darstellung des Commit-Verlaufs
- git diff** Darstellung von Änderungen

**git log** kann über die Taste q verlassen werden



# Einführung API

Als Teilnehmer soll ich am Ende dieser Übung...

- wissen, was eine Toolchain ist
- den Nutzen einer IDE in der Softwareentwicklung kennen
- die bereitgestellte virtuelle Maschine nutzen können
- einzeln mit einer lokalen Versionsverwaltung arbeiten können



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



Jetzt besteht die Möglichkeit das Sprintmeeting durchzuführen.

Protokolliert bitte

- die bearbeiteten Aufgaben der Vorwoche.
- die Zwischenstände der geplanten Aufgaben.
- was in der kommenden Woche bearbeitet werden soll.

Falls einige Punkte noch nicht bearbeitet werden konnten, kann das Meeting zu einem anderen Zeitpunkt durchgeführt werden.



# Literatur

- [1] S. Chacon und B. Straub, *Pro Git*. 2014.



# Ende

Vielen Dank für eure Aufmerksamkeit!

