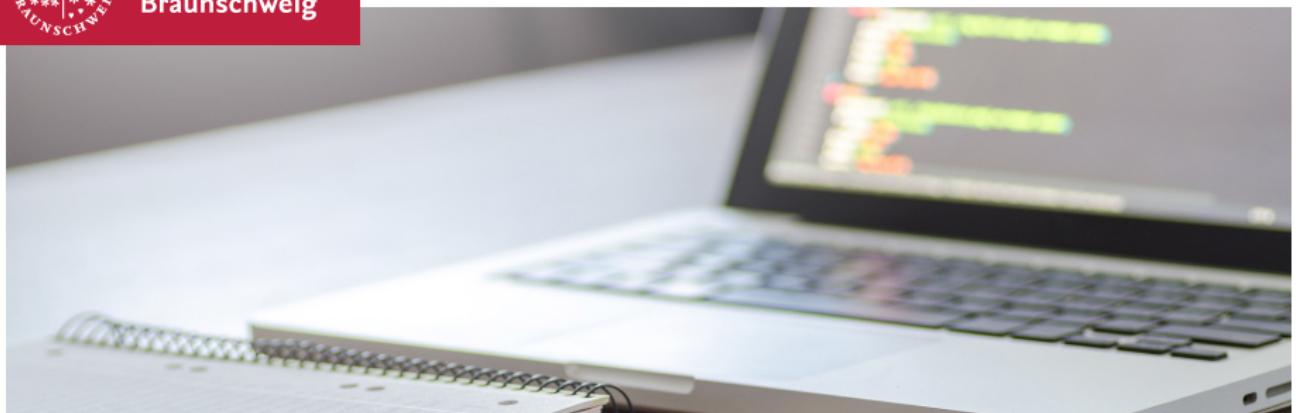




Technische  
Universität  
Braunschweig

Institut für  
Flugführung



# Entwicklungsumgebungen und Versionskontrolle

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,  
23. 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**



# Lehrziele

## Entwicklungsumgebungen und Versionskontrolle

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



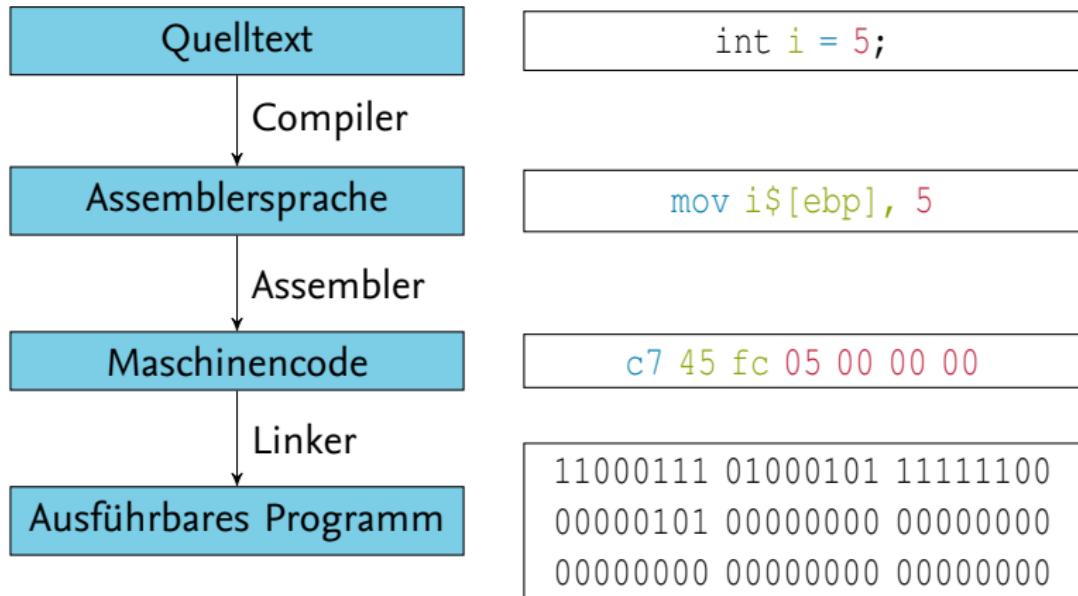
Die Toolchain

Entwicklungs-  
umgebungen

API-Tools

Projekt-  
erstellung

# Vom Quelltext zum ausführbaren Programm

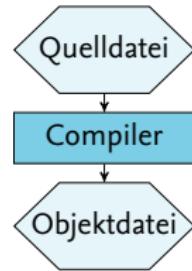


# Toolchain

## 1. KOMPILIEREN

Quelltext in Maschinencode  
umwandeln

→ Objektdatei

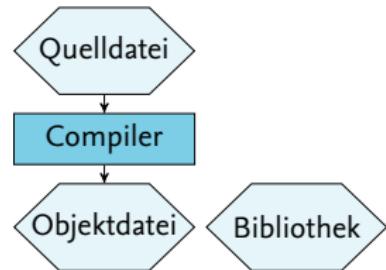


# Toolchain

## 1. KOMPILIEREN

Quelltext in Maschinencode  
umwandeln

→ Objektdatei



# Toolchain

## 1. KOMPILIEREN

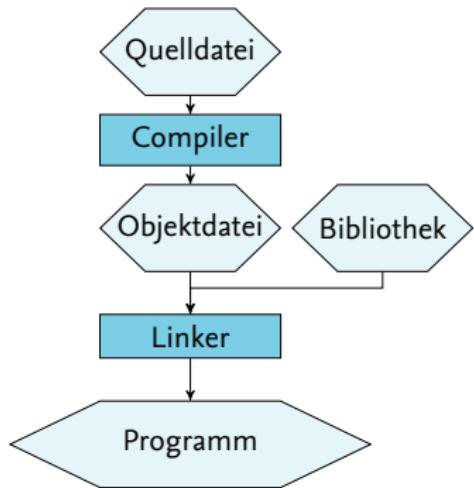
Quelltext in Maschinencode umwandeln

→ Objektdatei

## 2. LINKEN

Maschinencode und Bibliotheken zusammenfassen

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



# Toolchain

## 1. KOMPILIEREN

Quelltext in Maschinencode umwandeln

→ Objektdatei

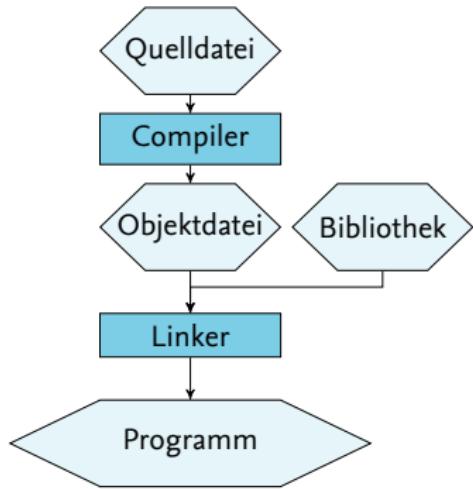
## 2. LINKEN

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



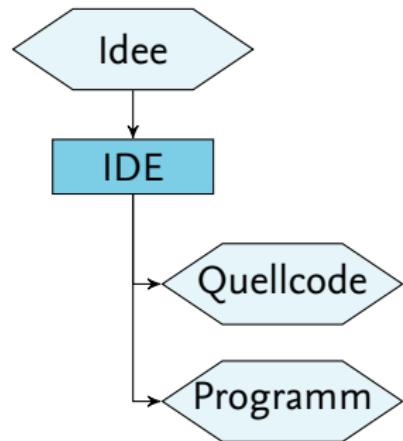
Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Toolchain?**



# 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 vollwärtige IDE  
→ Plug-Ins erforderlich
- Compiler müssen z. T. selbst installiert werden.

```

1 //include "main.h"
2
3 void setup() {
4     // LED output
5     for (int i = 0; i < 5; i++) {
6         digitalWrite(i, HIGH);
7     }
8
9     // Button Input
10    pinMode(2, INPUT_PULLUP);
11
12    // Initialize LCD
13    lcd.begin(16, 2);
14    lcd.createChar(0, SpecialChar);
15
16    void loop() {
17        if (digitalRead(2) == 0) {
18            lcd.setCursor(0, 0);
19            lcd.print("Hello World!");
20            lcd.setCursor(0, 1);
21            lcd.print("I'm a");
22            lcd.setCursor(0, 2);
23            lcd.print("LCD");
24            lcd.setCursor(0, 3);
25            delay(1000);
26        }
27
28        void printHelloWorld() {
29            long l = millis();
30            if (l >= m_lastUpdateTime + m_updateInterval) {
31                m_lastUpdateTime = l - (m_lastUpdateTime % 1);
32                m_lastUpdateTime += 1;
33            }
34        }
35        if (!availableSerial) {
36            availablePort();
37        }
38        if (availablePort) {
39            availableSerial = false;
40            lcd.begin(16, 2);
41            lcd.createChar(0, SpecialChar);
42            lcd.setCursor(0, 0);
43            lcd.print("Hello World!");
44            lcd.setCursor(0, 1);
45            lcd.print("I'm a");
46            lcd.setCursor(0, 2);
47            lcd.print("LCD");
48            lcd.setCursor(0, 3);
49            lcd.print("I2C");
50            lcd.setCursor(0, 4);
51            lcd.print("I2S");
52            lcd.setCursor(0, 5);
53            lcd.print("SPI");
54            lcd.setCursor(0, 6);
55            lcd.print("PS2");
56            lcd.setCursor(0, 7);
57            lcd.print("DS18B20");
58            lcd.setCursor(0, 8);
59            lcd.print("DHT");
56        }
57    }
58
59    case 1:
60        lcd.print("Hello World!");
61        runID_Sequential();
62        break;
63    case 2:
64        lcd.print("Hello World!");
65        runID_Random();
66        break;
67    case 3:
68        lcd.print("Hello World!");
69        runID_Hostname();
70        break;
71    }
72}

```

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



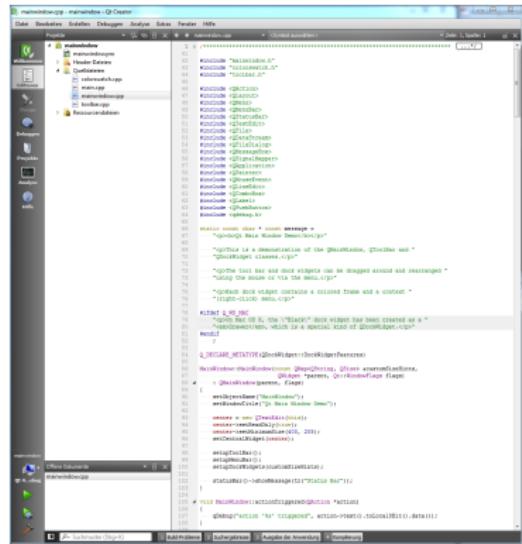
# Qt Creator



## 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 14. 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

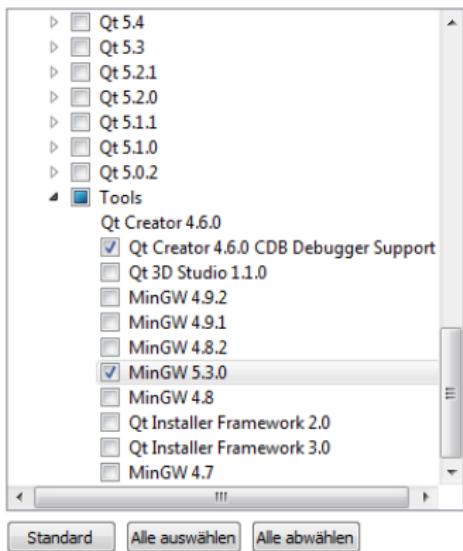
# Qt & Qt CREATOR

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

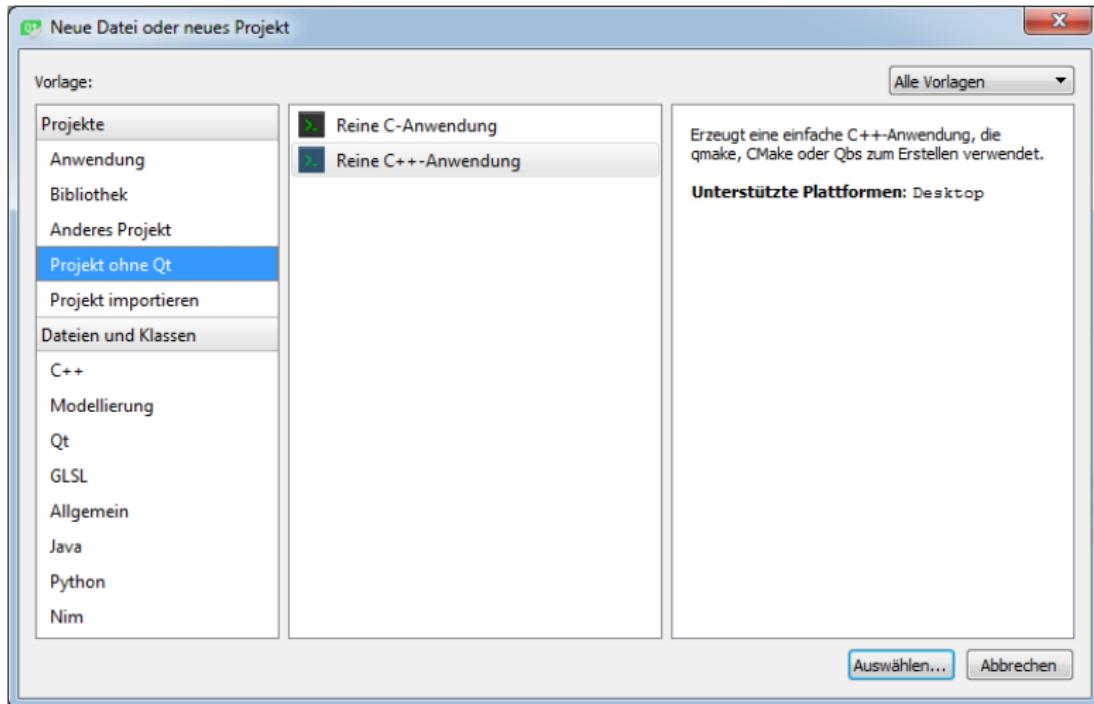
## Komponenten auswählen

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



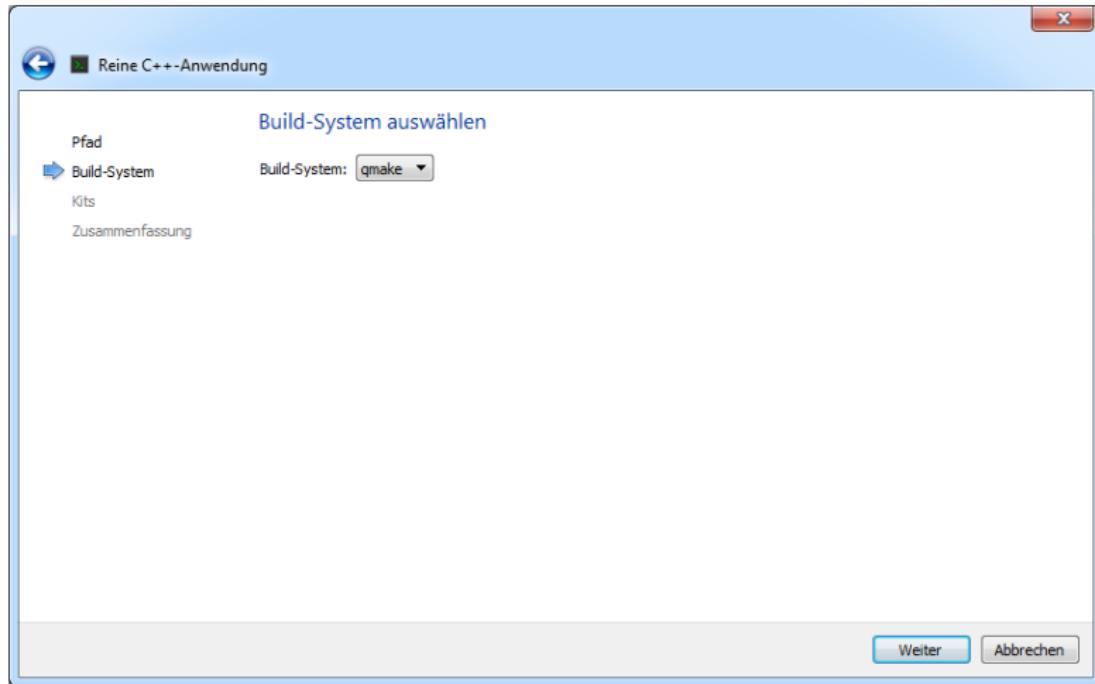
# Beispiel

# Erstes C++-Projekt



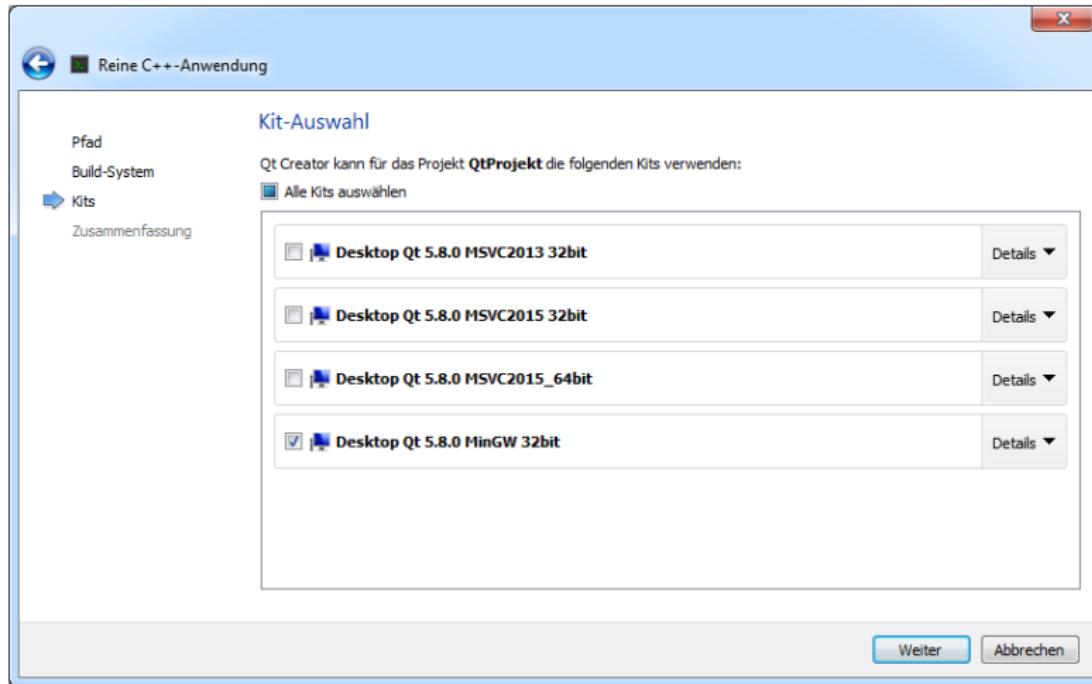
# Beispiel

# Erstes C++-Projekt



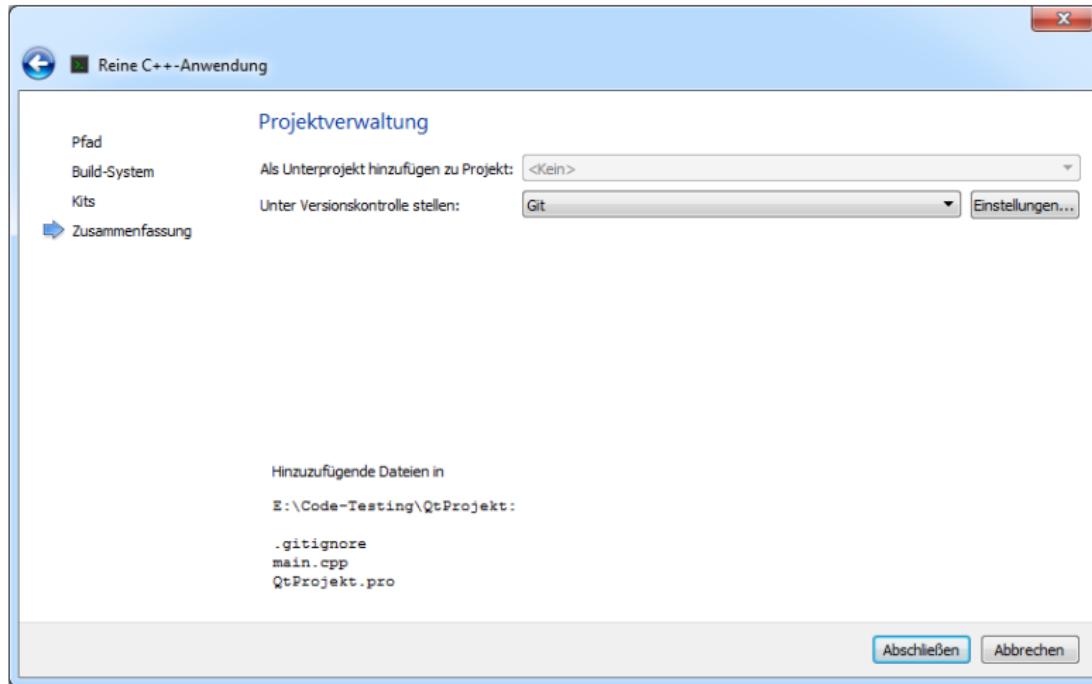
# Beispiel

# Erstes C++-Projekt



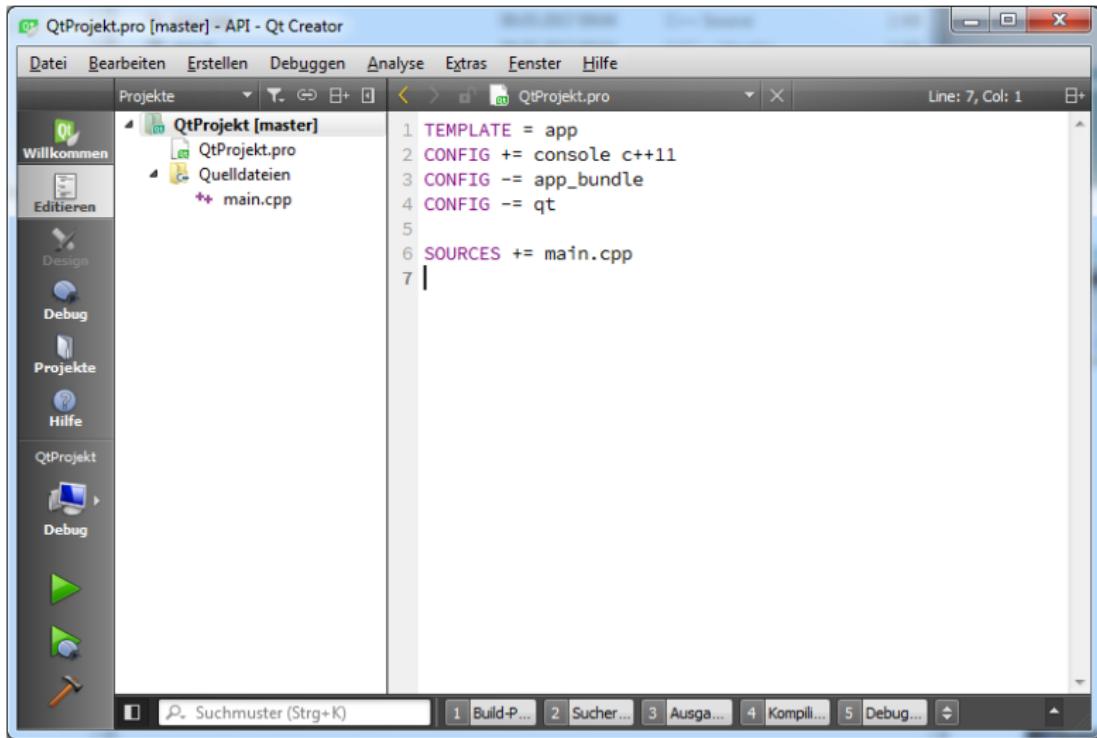
# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt



```
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 interface with the following details:

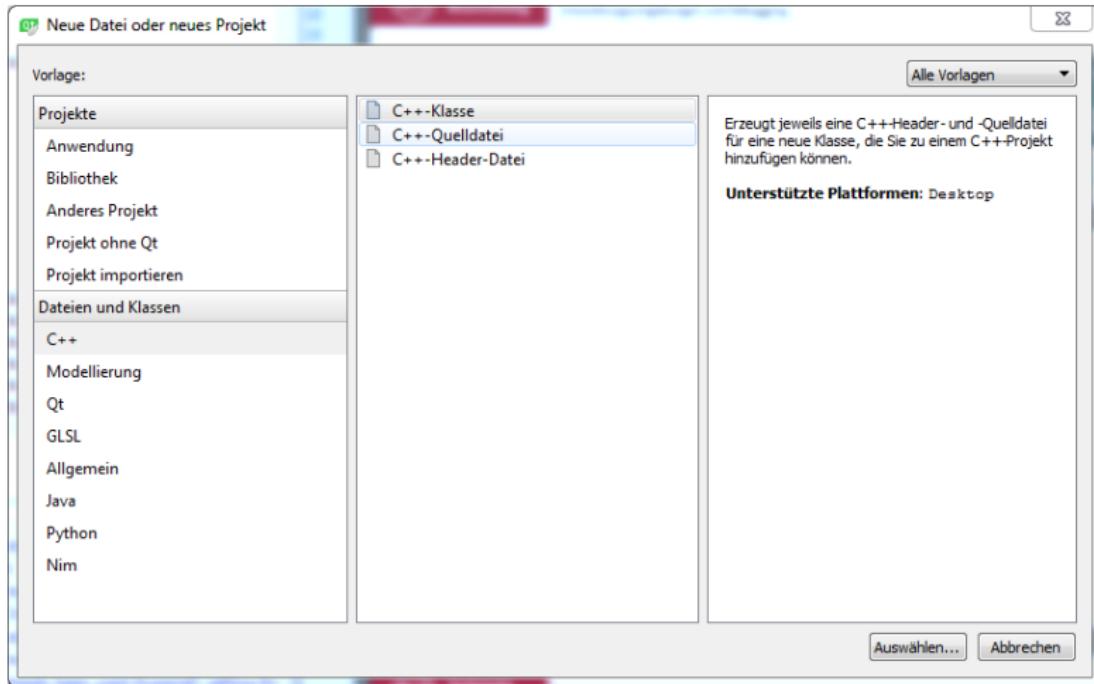
- Title Bar:** main.cpp [master] - API - Qt Creator
- Menu Bar:** Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, Hilfe
- Toolbars:** Willkommen, Editieren, Design, Debug, Projekte, Hilfe
- Project Tree:** QtProjekt [master] (QtProjekt.pro, Quelldateien, main.cpp)
- Code Editor:** main.cpp (containing the provided C++ code)
- Bottom Bar:** Suchmuster (Strg+K), Build-P..., Sucher..., Ausga..., Kompili..., Debug...

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "Hello World!" << endl;
    return 0;
}
```



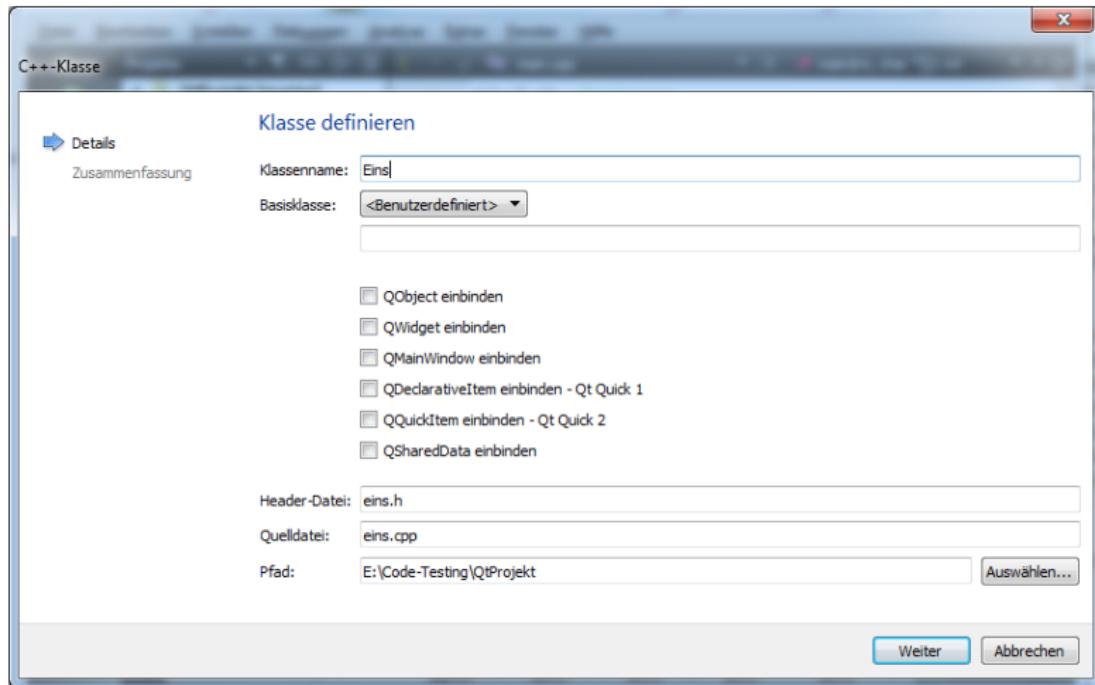
# Beispiel

# Erstes C++-Projekt



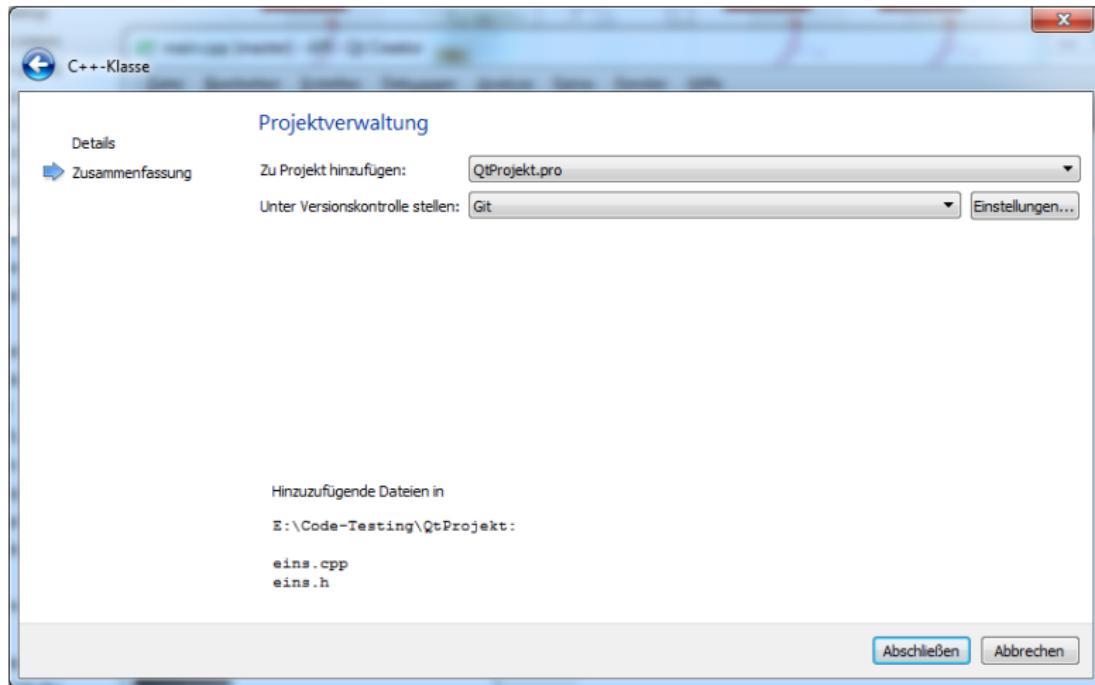
# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt



# Beispiel

# Erstes C++-Projekt

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

- Title Bar:** QtProjekt.pro [master] - API - Qt Creator
- Menu Bar:** Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, Hilfe
- Toolbars:** Willkommen, Editieren, Design, Debug, Projekte, Hilfe
- Project Tree:** QtProjekt [master] (selected), QtProjekt.pro, Header-Dateien (eins.h), Quelldateien (eins.cpp, main.cpp)
- Code Editor:** QtProjekt.pro (content shown below)
- Code 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
```
- Bottom Bar:** Suchmuster (Strg+K), Build-P..., Sucher..., Ausga..., Kompili..., Debug...



# Beispiel

# Erstes C++-Projekt

The screenshot shows the Qt Creator IDE interface. The title bar reads "eins.h [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. Under Projekte, there is a "QtProjekt [master]" entry with sub-options for QtProjekt.pro and Header-Dateien. The Header-Dateien section contains "eins.h". Below it is a Quelldateien section with "eins.cpp" and "main.cpp". The main central area displays the content of "eins.h":

```
1 ifndef EINS_H
2 define EINS_H
3
4
5 class Eins
6 {
7 public:
8     Eins();
9 };
10
11 endif // EINS_H
```

At the bottom, there is a search bar labeled "Suchmuster (Strg+K)" and several tabs labeled 1 Build-P..., 2 Sucher..., 3 Ausga..., 4 Kompili..., 5 Debug... .



Gibt es Fragen oder Anmerkungen zu dem Thema  
IDEs?



# Abgehakt

## Entwicklungsumgebungen und Versionskontrolle

Als Teilnehmer soll ich am Ende dieser Übung...

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



git



Motivation

git

Übung



# Versionskontrolle?

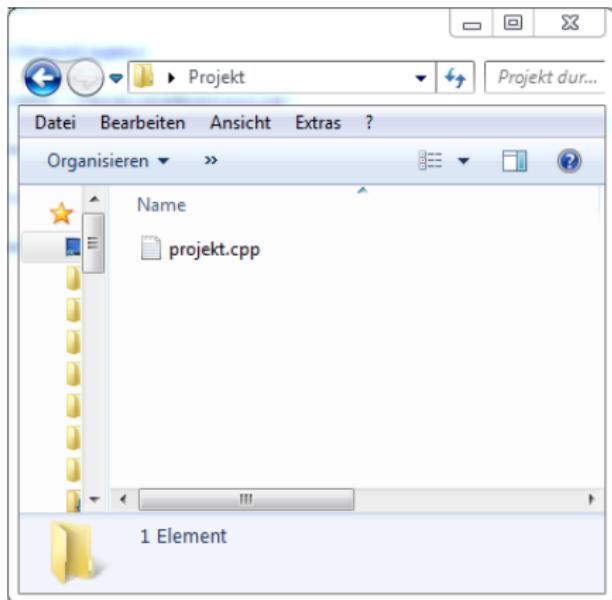


Abbildung 1: Versionsverwaltung durch Copy und Paste



# Versionskontrolle?

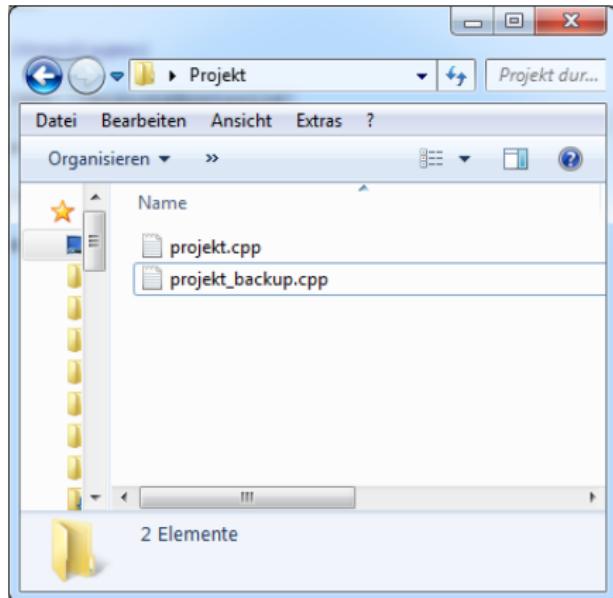


Abbildung 1: Versionsverwaltung durch Copy und Paste

# Versionskontrolle?

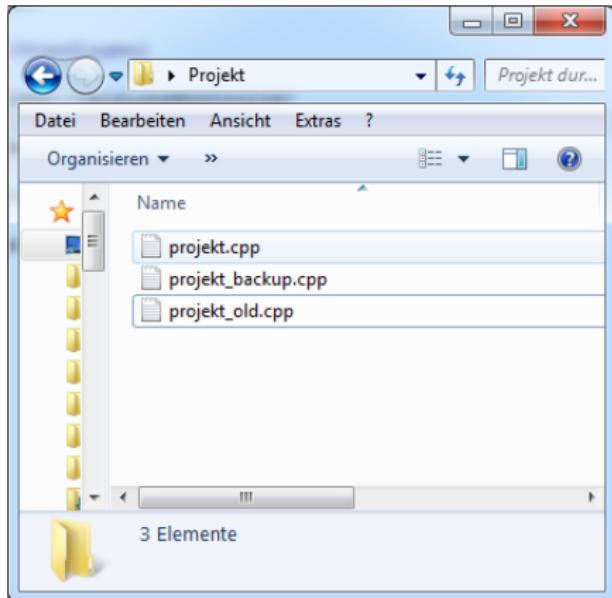


Abbildung 1: Versionsverwaltung durch Copy und Paste



# Versionskontrolle?

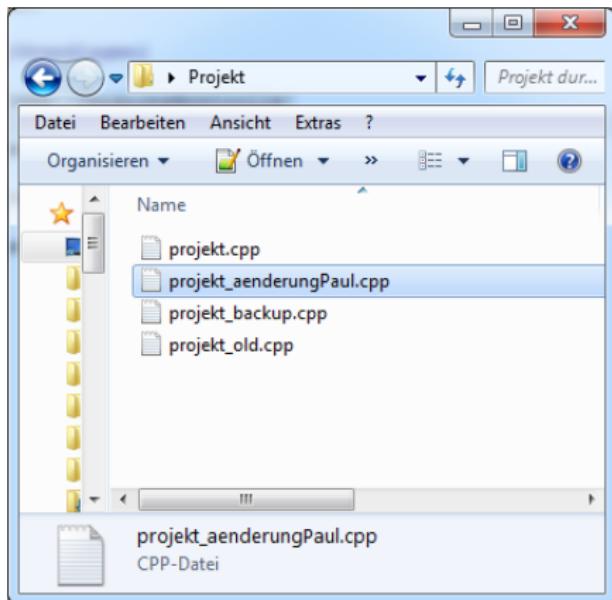


Abbildung 1: Versionsverwaltung durch Copy und Paste

# Versionskontrolle?

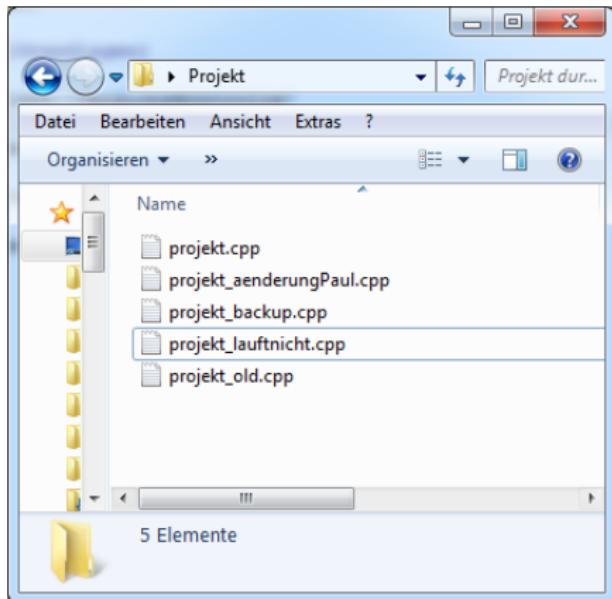


Abbildung 1: Versionsverwaltung durch Copy und Paste

# Versionskontrolle?

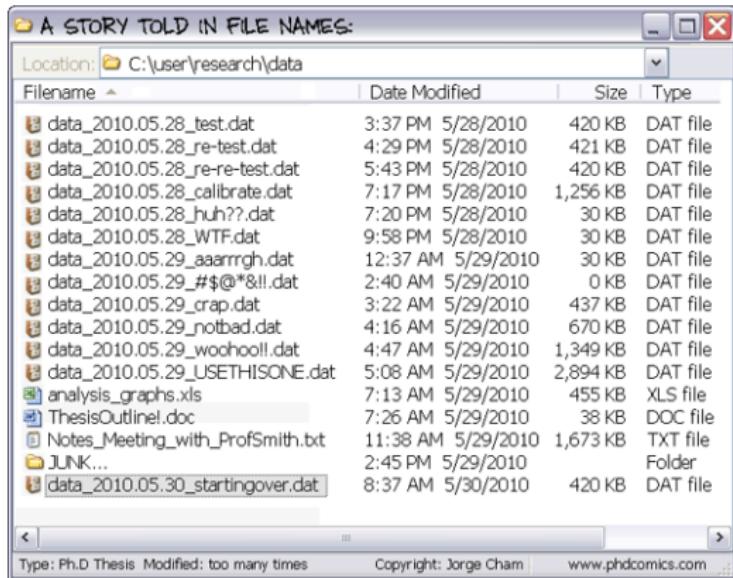


Abbildung 2: Versionierung mit Dateinamen  
“Piled Higher and Deeper” by Jorge Cham  
<http://www.phdcomics.com/comics.php?f=1323>

# 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

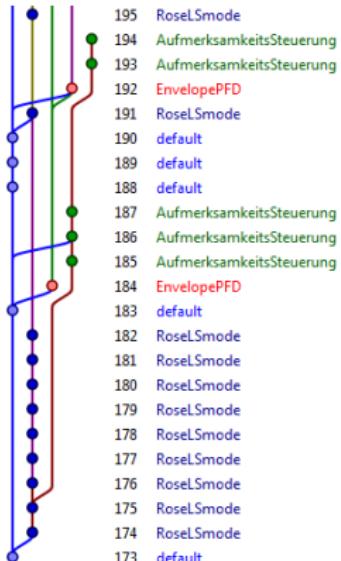


Abbildung 3: Versionsbaum

# 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

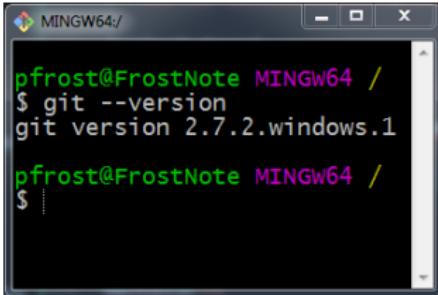
git ist ein Kommandozeilenprogramm und kann über Argumente gesteuert werden.

## Vorteile:

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

## Nachteile:

- Bedienung ist nicht intuitiv



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

Link:

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



# Verwendung von git über GITKRAKEN

GITKRAKEN liefert eine graphische Oberfläche für die Verwendung von git.

## Vorteile:

- git wird automatisch mitinstalliert
- Einfache Bedienung



## Nachteile:

- Bei kommerzieller Verwendung kostenpflichtig

## Link:

<https://www.gitkraken.com/git-client>



# 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/TUBSAPISS2019/API-Materialien/  
Aufgaben/03-Entwicklungstools/loesungen.zip](https://github.com/TUBSAPISS2019/API-Materialien/Aufgaben/03-Entwicklungstools/loesungen.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"
```

## Wichtig

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





# git config

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

The screenshot shows the 'My Profiles' section of the GitKraken application. At the top, there is a checkbox labeled 'Keep my .gitconfig updated with my GitKraken Profile preferences' which is checked. Below the checkbox, a descriptive text states: 'When checked, GitKraken will update your global .gitconfig to match GitKraken Preferences, including commit author name and email, and any GPG settings.' A green button labeled 'Add A Profile' is located on the left. In the main list area, there is one profile entry for 'Paul' with the details: 'Paul Frost' and 'p.frost@tu-braunschweig.de'. To the right of this entry is a blue 'Edit Profile' button. The background of the window is dark grey.

Abbildung 4: GITKRAKEN Profileinstellung

## Wichtig

Die E-Mail-Adresse sollte identisch mit der 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

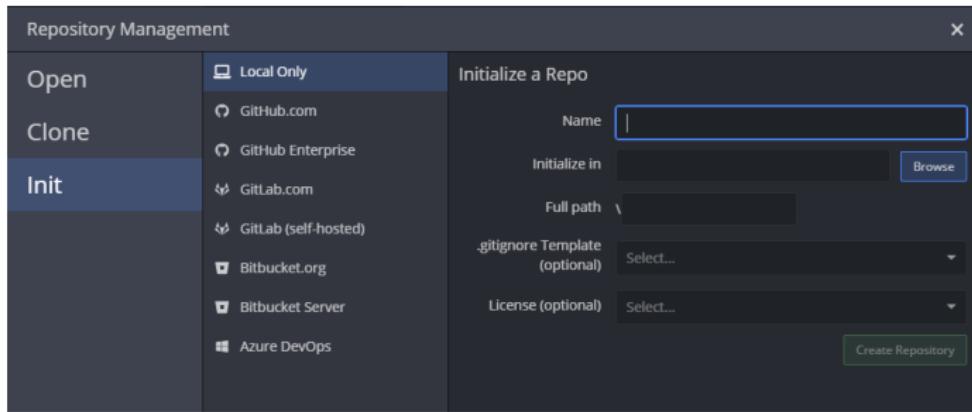


Abbildung 5: GITKRAKEN Repository erstellen (File→Init Repo)



# .git

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

- Der .git-Ordner ist das eigentliche Repository
- Sämtliche Revisionen sind in diesem Ordner
- Das Repository kann nur wachsen

## Wichtig

Der .git-Ordner darf nicht gelöscht werden!

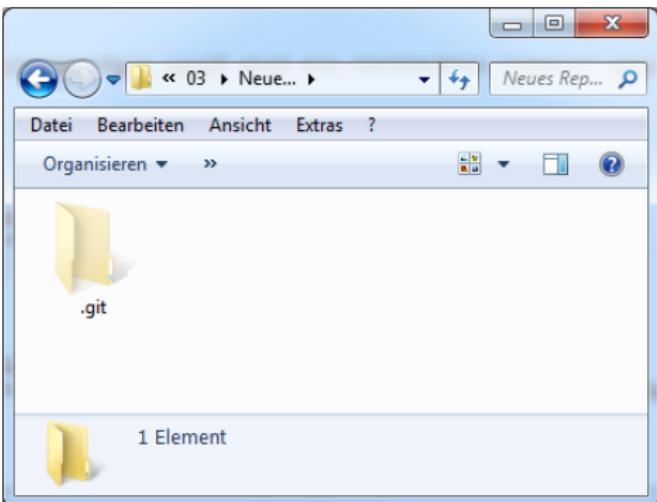


Abbildung 6: Arbeitsverzeichnis

# git Arbeitsfluss

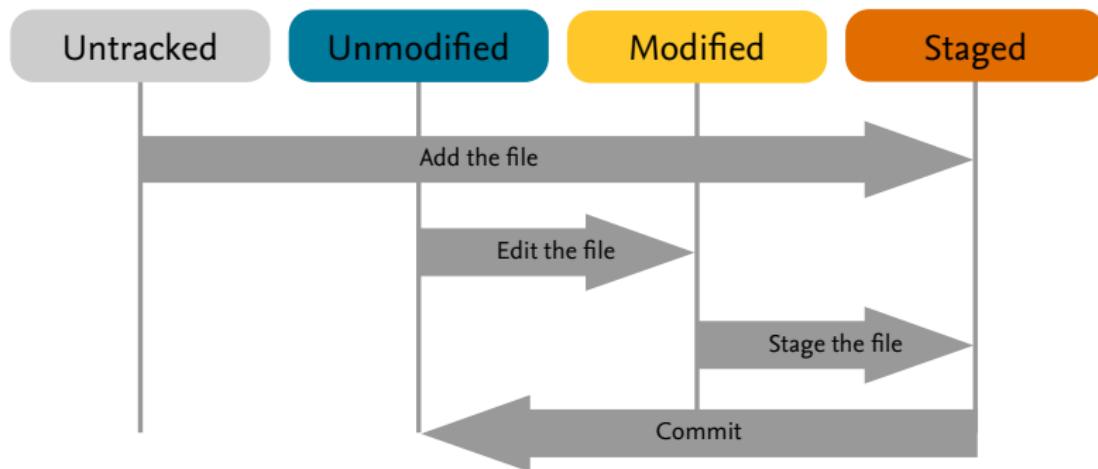
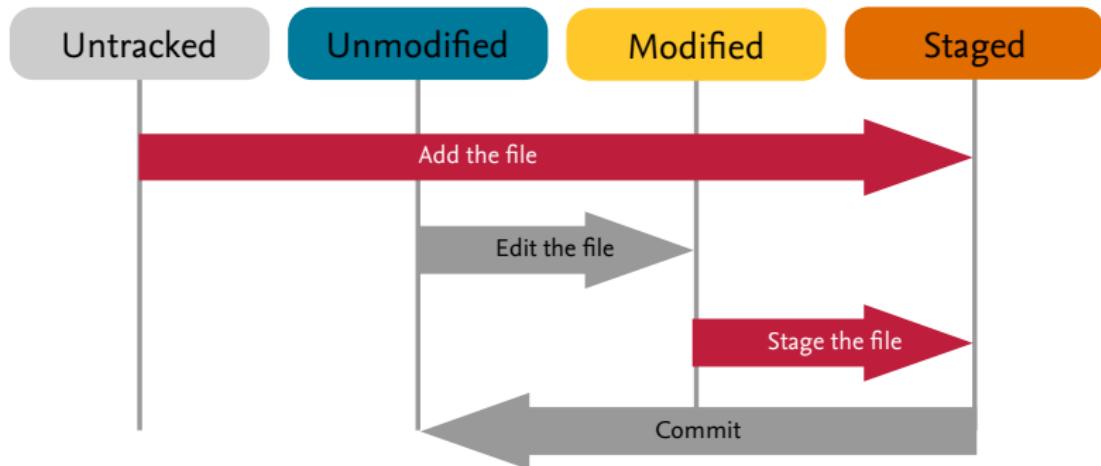


Abbildung 7: git workflow [1]



# 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 9: 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 add



Im rechten Dock des Fensters können die Änderungen eingesehen und in die Staged-Ebene hinzugefügt werden.

10 file changes on Entwurf

Unstaged Files (10)

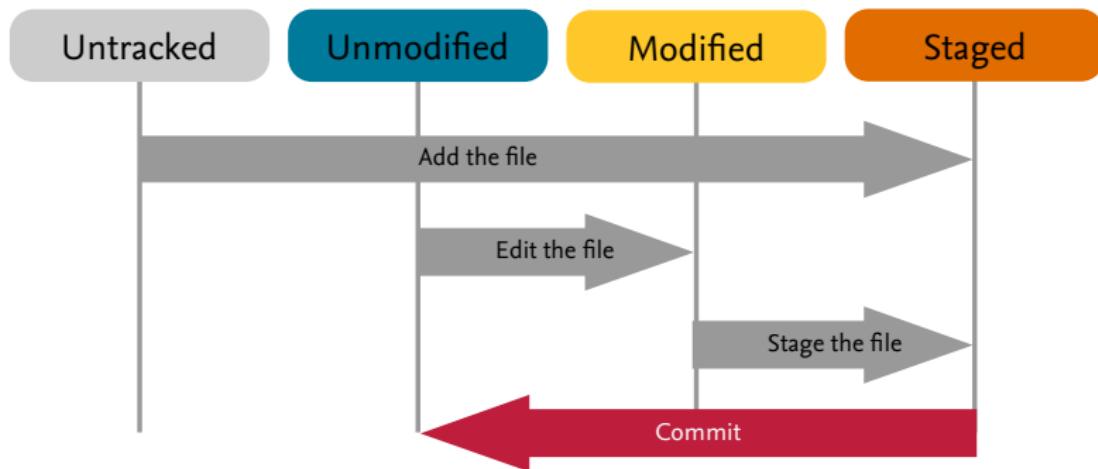
Stage all changes

- 03-IDEs-git/t02/git.tex
- 03-IDEs-git/t02/img/gallery-keif.jpg
- 03-IDEs-git/t02/img/git-newrepo.png
- 03-IDEs-git/t02/img/github-config.png



# git Arbeitsfluss

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





# git commit

Ein **commit** sichert den aktuellen Stand in den aktiven Zweig

Listing 11: Daten

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

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

## Wichtig

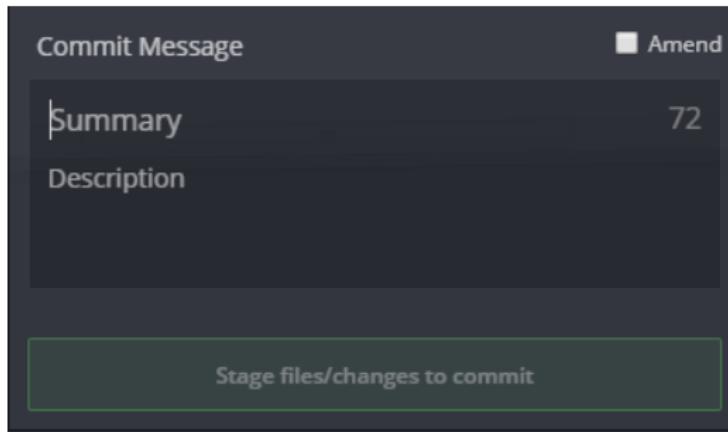
Die Commit-Nachricht ist bewertungsrelevant!



# git commit



Ein **commit** sichert den aktuellen Stand in den aktiven Zweig



## Wichtig

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]



# Zweige

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

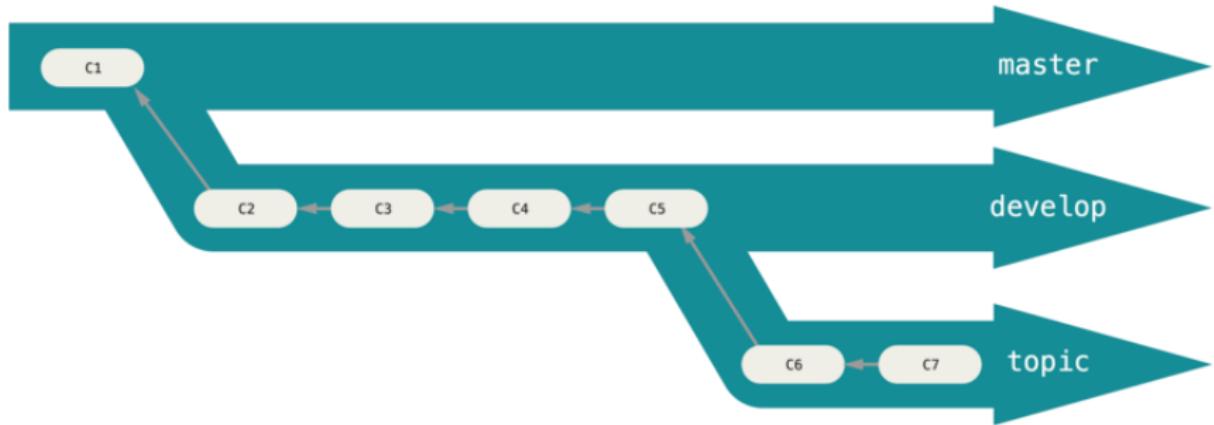


Abbildung 8: Zweige in git [1]



# git checkout

Listing 13: Neuen Zweig erstellen und in diesen Zweig wechseln

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

Listing 14: Zu einem existierenden Zweig wechseln

```
git checkout "Name_bestehender_Zweig"
```

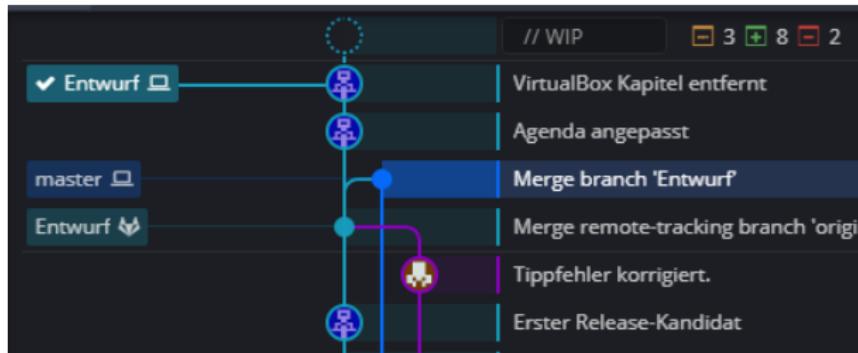
Alle Änderungen an bereits versionierten Dateien müssen vorher mit einem Commit gesichert werden.





# git checkout

Über einen Doppelklick auf ein Zweig-Label kann in einen anderen Zweig gewechselt werden.



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



# Situation

Ich habe mich vertan...



# Unversionierte Änderungen rückgängig machen



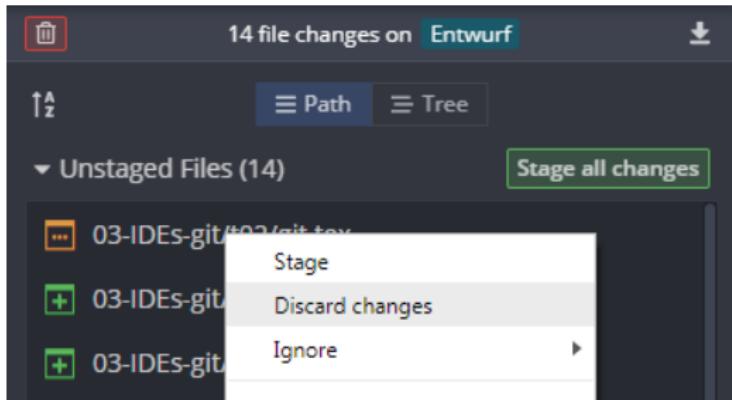
Der Befehl **git checkout** kann gleichzeitig dafür verwendet werden, Änderungen bis zu dem letzten Commit wiederherzustellen. Hierzu wird die Option **--** als Option vor einer Liste mit Dateien gesetzt.

Listing 17: 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.

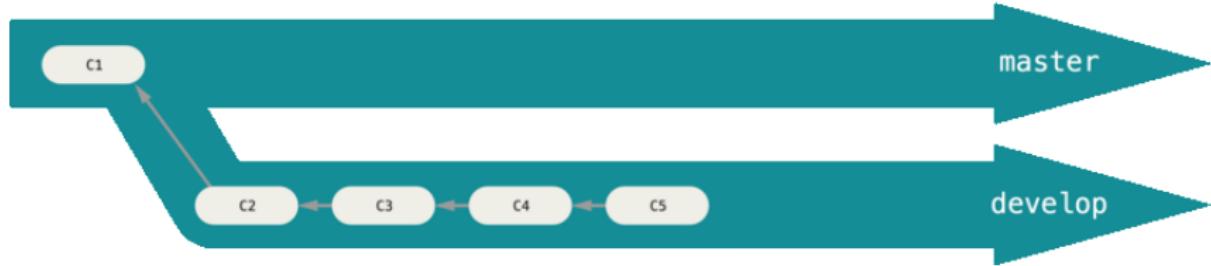


Abbildung 9: Vor einem Merge vgl. [1]

# Zusammenführen von Inhalten

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

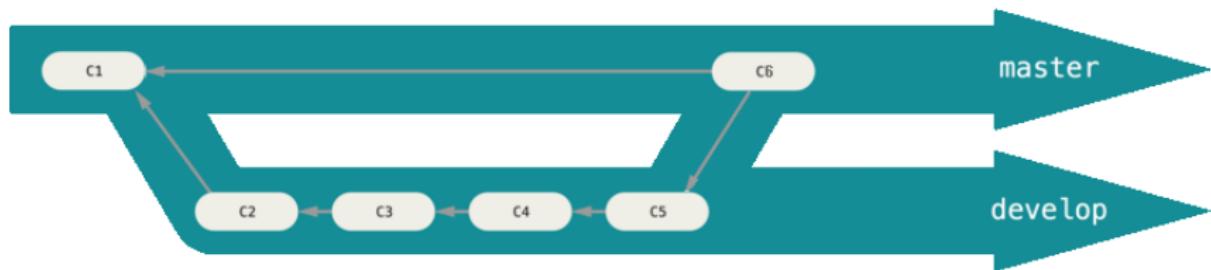


Abbildung 9: Nach einem Merge vgl. [1]



# git merge



Listing 19: Zusammenführen von Zweigen

```
git merge "Name_Zweig"
```

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



# git merge



Das Zusammenführen zweier Zweige erfolgt über das Ziehen eines Zweig-Labels auf ein anderes Zweig-Label.

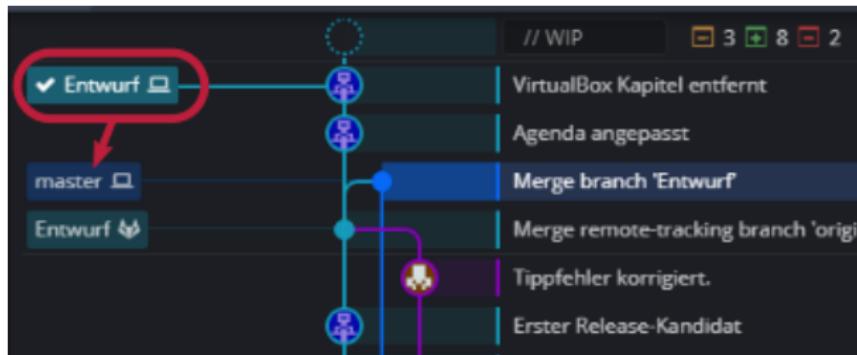


Abbildung 10: Zusammenführen des Entwurf-Zweigs in den master-Zweig



# Ignore-Datei

# Dateien ignorieren

Listing 21: 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



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



# Abgehakt

## Entwicklungsumgebungen und Versionskontrolle

Als Teilnehmer soll ich am Ende dieser Übung...

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



# Literatur

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



# Ende

Vielen Dank für eure Aufmerksamkeit!

