Wie Git denn das?
Einführung in die Versionskontrolle mit Git

Tim Pollandt

Fachschaft Informatik TU Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

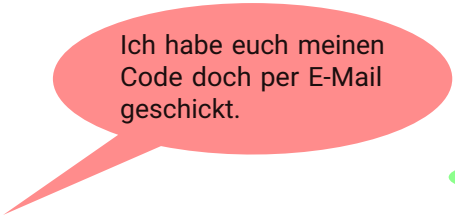
Warum Git?

Funktionsweise
- Normalfall
- Abweichungen vom Normalfall
- Zusammenfassung
- Branching

Best Practices

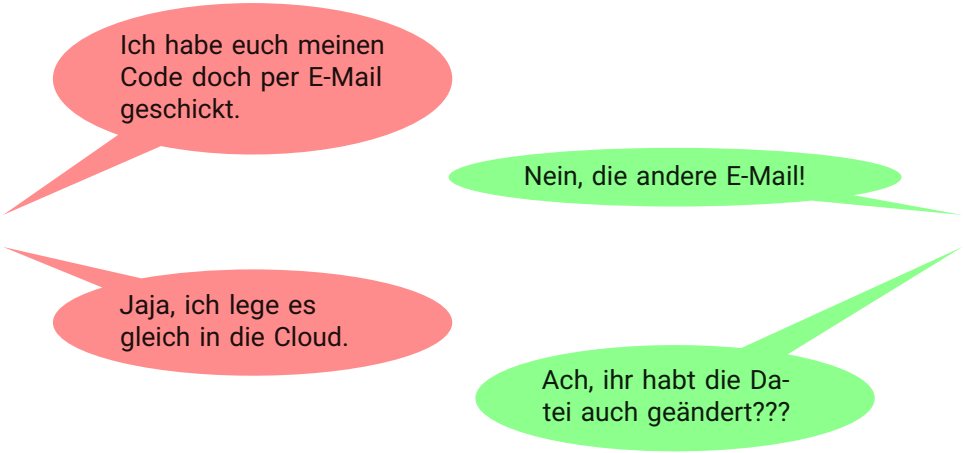
Warum Quellcodeverwaltung?

TECHNISCHE UNIVERSITÄT DARMSTADT



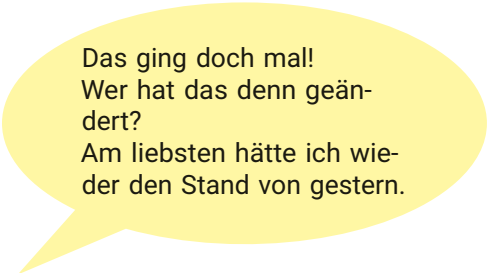
Warum Quellcodeverwaltung?

TECHNISCHE UNIVERSITÄT DARMSTADT



Warum Quellcodeverwaltung?

TECHNISCHE UNIVERSITÄT DARMSTADT



Darum Quellcodeverwaltung!

TECHNISCHE UNIVERSITÄT DARMSTADT

- zentrales Quellcodeverzeichnis
- offline (und online)
- gleichzeitige Bearbeitung: Lösung von Versionskonflikten
- Versionshistorie (Autor, Zeit)

Insgesamt:

Vereinfachte Zusammenarbeit

Warum Git?

TECHNISCHE UNIVERSITÄT DARMSTADT

- weit verbreitet
- die im *Open Source*-Bereich oft genutzten Plattformen **GitLab** und **GitHub** basieren auf Git
- auf https://git.rwth-aachen.de verfügbar
- auf eigenen Servern leicht zu hosten
- Alternativen sind zum Beispiel **Mercurial** und **SVN**

Funktionsweise

TECHNISCHE UNIVERSITÄT DARMSTADT

1f42b40 **Initial commit**

```
+   int add(int a, int b){
+   // TODO implement
+   }
+   int sub(int a, int b){
+   // TODO implement
+   }
```

Funktionsweise

TECHNISCHE UNIVERSITÄT DARMSTADT

1f42b40 **Initial commit**

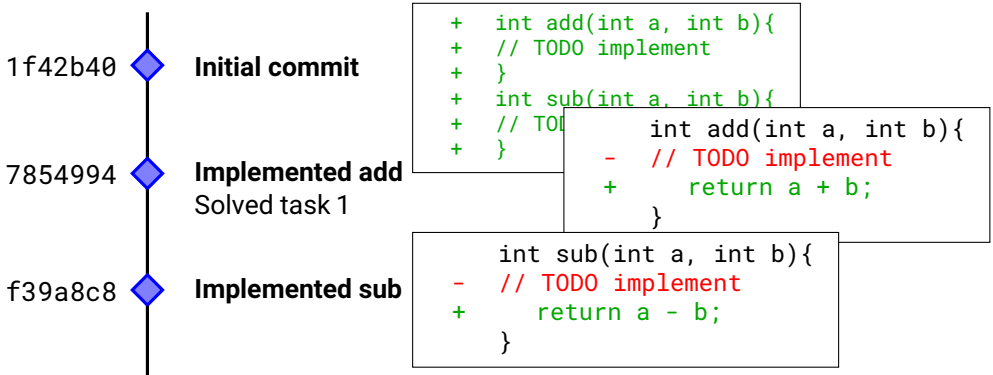
```
+   int add(int a, int b){
+   // TODO implement
+   }
+   int sub(int a, int b){
+   // TOD
+   }
```

7854994 **Implemented add**
Solved task 1

```
    int add(int a, int b){
-   // TODO implement
+       return a + b;
    }
```

Funktionsweise

TECHNISCHE UNIVERSITÄT DARMSTADT



Funktionsweise

TECHNISCHE UNIVERSITÄT DARMSTADT

1f42b40 **Initial commit**

```
+ int add(int a, int b){
+ // TODO implement
+ }
+ int sub(int a, int b){
+ // TOD
+ }
```

7854994 **Implemented add**

Solved task 1

```
    int add(int a, int b){
-   // TODO implement
+       return a + b;
    }
```

f39a8c8 **Implemented sub**

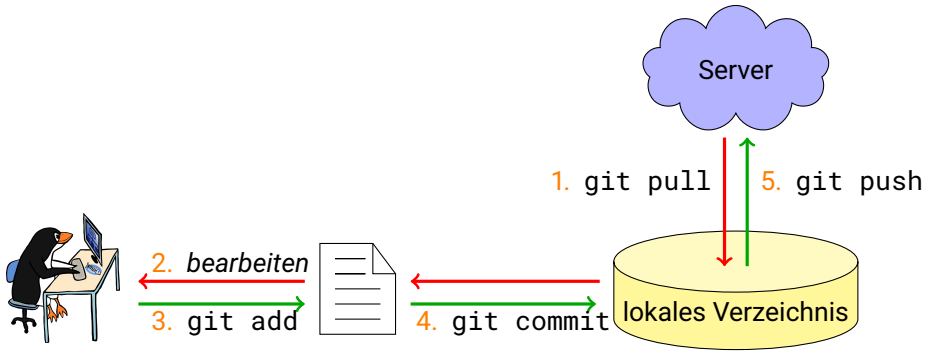
```
    int sub(int a, int b){
-   // TODO implement
+       return a - b;
    }
```

54cdb2c **Revert: Implemented add**

We do not need to implement this for the first assignment.

```
    int add(int a, int b){
+   // TODO implement
-       return a + b;
    }
```

Schritte in Git (Normalfall)

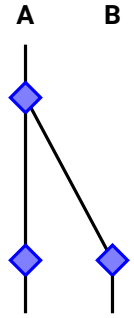


Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

```
+   int add(int a, int b){
+   // TODO implement
+   }
+   int sub(int a, int b){
+   // TODO implement
+   }
```

```
    int sub(int a, int b){
-   // TODO implement
+      return a - b;
    }
```



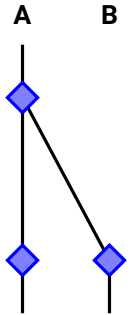

```
    int add(int a, int b){
-   // TODO implement
+      return a + b;
    }
```

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int add(int a, int b){
// TODO implement
}
int sub(int a, int b){
// TODO implement
}
```

```
int add(int a, int b){
// TODO implement
}
int sub(int a, int b){
   return a - b;
}
```



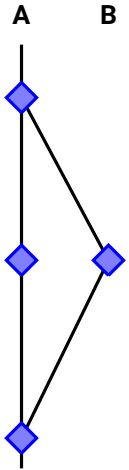

```
int add(int a, int b){
   return a + b;
}
int sub(int a, int b){
// TODO implement
}
```

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int add(int a, int b){
// TODO implement
}
int sub(int a, int b){
// TODO implement
}
```

```
int add(int a, int b){
// TODO implement
}
int sub(int a, int b){
   return a - b;
}
```




```
int add(int a, int b){
   return a + b;
}
int sub(int a, int b){
// TODO implement
}
```

```
int add(int a, int b){
   return a + b;
}
int sub(int a, int b){
   return a - b;
}
```

Versionen zusammenführen (*merge*)

TECHNISCHE UNIVERSITÄT DARMSTADT

Reihenfolge ist wichtig!
Wenn das Remote-Repository zwischendurch geändert wurde:

1. `git pull`:
   Änderungen vom Server empfangen und zusammenführen
2. `git push`:
   Die zusammengeführte Version auf den Server übertragen

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir nehmen den folgenden Fall an:

1. **Alice** lädt Änderungen auf Computer **A** herunter (*git pull*).

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir nehmen den folgenden Fall an:

1. **Alice** lädt Änderungen auf Computer **A** herunter (*git pull*).
2. **Bob** lädt Änderungen auf Computer **B** herunter (*git pull*).

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir nehmen den folgenden Fall an:

1. **Alice** lädt Änderungen auf Computer **A** herunter (*git pull*).
2. **Bob** lädt Änderungen auf Computer **B** herunter (*git pull*).
3. **Alice** und **Bob** bearbeiten eine Datei und committen die jeweilige Änderung.

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir nehmen den folgenden Fall an:

1. **Alice** lädt Änderungen auf Computer **A** herunter (*git pull*).
2. **Bob** lädt Änderungen auf Computer **B** herunter (*git pull*).
3. **Alice** und **Bob** bearbeiten eine Datei und committen die jeweilige Änderung.
4. **Alice** lädt ihre Änderungen hoch (*git push*).

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir nehmen den folgenden Fall an:

1. **Alice** lädt Änderungen auf Computer **A** herunter (*git pull*).
2. **Bob** lädt Änderungen auf Computer **B** herunter (*git pull*).
3. **Alice** und **Bob** bearbeiten eine Datei und committen die jeweilige Änderung.
4. **Alice** lädt ihre Änderungen hoch (*git push*).
5. **Bob** versucht seine Änderungen hochzuladen (*git push*).

Gleichzeitiges Bearbeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir nehmen den folgenden Fall an:

1. **Alice** lädt Änderungen auf Computer **A** herunter (*git pull*).
2. **Bob** lädt Änderungen auf Computer **B** herunter (*git pull*).
3. **Alice** und **Bob** bearbeiten eine Datei und committen die jeweilige Änderung.
4. **Alice** lädt ihre Änderungen hoch (*git push*).
5. **Bob** versucht seine Änderungen hochzuladen (*git push*).

!! **Error**: beide Verzeichnisse wurden bearbeitet.

Da Bobs Änderung noch auf der Version vor Alice' Änderung basiert,
kann sie nicht einfach übertragen werden (um nichts zu überschreiben).

- **Lösung:** Versionen lokal zusammenführen

Merge-Konflikt

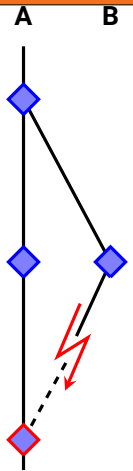
TECHNISCHE UNIVERSITÄT DARMSTADT

- Änderungen in der gleichen Zeile gleicher Datei
  $\Rightarrow$ Git kann Versionen nicht automatisch zusammenführen
- manuell *Merge-Konflikt* beheben

Merge-Konflikt

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int add(int a, int b){
// TODO implement
}
int sub(int a, int b){
// TODO implement
}
```




```
int add(int a, int b){
// TODO implement
}
int sub(int a, int b){
   return a - b;
}
```

```
int add(int a, int b){
   return a + b;
}
int sub(int a, int b){
   return a + -1*b;
}
```

Merge-Konflikt

TECHNISCHE UNIVERSITÄT DARMSTADT

- bei Änderungen in der gleichen Zeile der gleichen Datei kann Git die Versionen nicht automatisch zusammenführen

→ Merge-Konflikt

  - wieder erst *pullen*
  - Merge-Konflikt manuell in Datei beheben:

```
1  int sub(int a, int b){
2  <<<<<<< HEAD
3      return a + -1*b;
4  =======
5      return a - b;
6  >>>>>>> e41aced...
7  }
```

  - *committen* und Commit-Nachrichten **Merge branch 'main' of...** ändern
  - *pushen*

Rezept

TECHNISCHE UNIVERSITÄT DARMSTADT

1. `git pull`
2. *Dateien ändern* und mit `git add` übernehmen
3. `git commit`
4. `git push`
   $\Rightarrow$ Konflikt: `git pull` $\rightarrow$ *Konflikte beheben* $\rightarrow$ Schritt 3.

Problem: unfertige Features

TECHNISCHE UNIVERSITÄT DARMSTADT

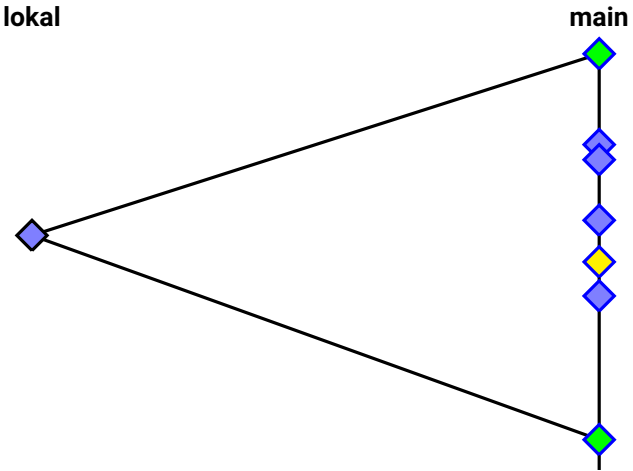
- Funktionen sind noch unfertig
- Funktionen werden verworfen
- jederzeit funktionierender Stand benötigt

Wir wollen jederzeit kollaborativ an allem arbeiten, aber trotzdem nur fertige Funktionen in der aktuellen Version.

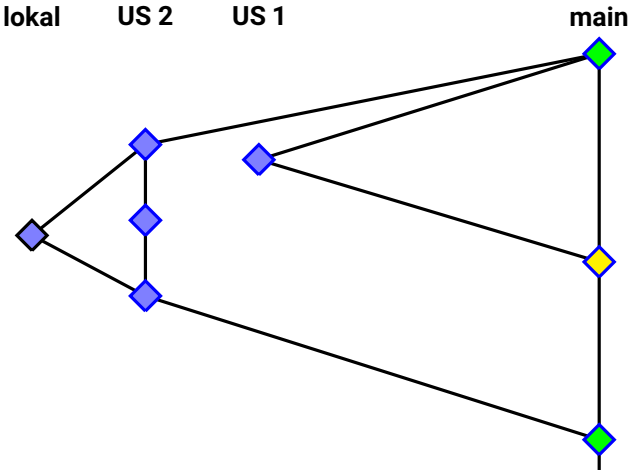
- **Lösung: Branching**

Branching

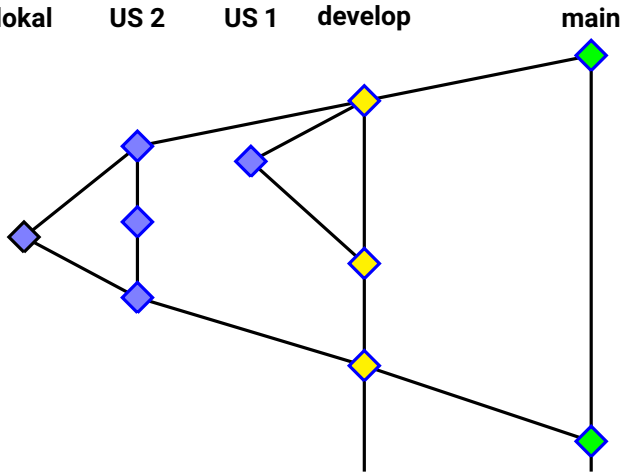
TECHNISCHE UNIVERSITÄT DARMSTADT



Branching – `feature`-Branches

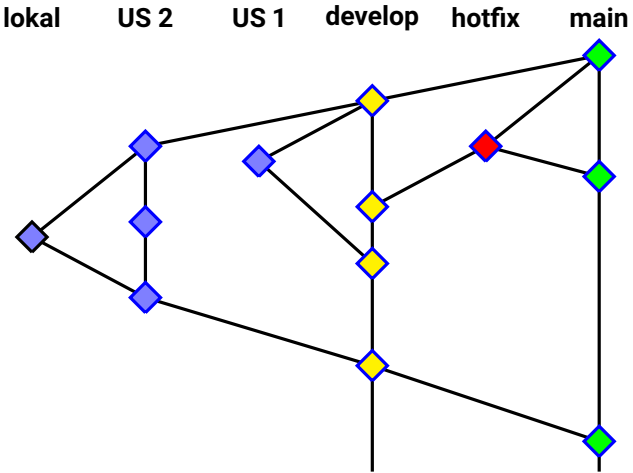


Branching – develop-Branch



Branching – Hotfixes

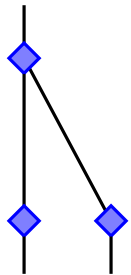
TECHNISCHE UNIVERSITÄT DARMSTADT



Branching

TECHNISCHE UNIVERSITÄT DARMSTADT

Commit:
- Commit-ID
- Parent-Commit-ID
- Autor <E-Mail>
- Committer <E-Mail>
- Datum
- Diff



Branch:
- Bezeichnung
- Commit-ID

Branching – Befehle

TECHNISCHE UNIVERSITÄT DARMSTADT

- `git branch` zeigt alle verfügbaren Branches
- `git branch branchname` erstellt den Branch „branchname“
- `git checkout branchname` wechselt zum Branch „branchname“
- `git merge branchname` merged Branch „branchname“ in aktuellen Branch

.gitignore-Datei

TECHNISCHE UNIVERSITÄT DARMSTADT

Mithilfe von `.gitignore`-Dateien, kann man festlegen, welche Dateien (nicht) von Git verwaltet werden sollen.

```
1  # kompilierte Java-Dateien ausschliessen
2  *.class
3
4  # Export-Daten nicht synchronisieren
5  exportiert/
6  # Datei, die doch synchronisiert werden soll
7  !exportiert/manuellErstellteDatei.tpt
8
9  # eine bestimmte ausgeschlossene Datei
0  zeugs/nichtFuerAndereRelevantes.tpt
```

How to git gud!

TECHNISCHE UNIVERSITÄT DARMSTADT

- sinnvolle Commit-Nachrichten schreiben
- Binärdateien u. Ä. in `.gitignore`-Datei aufnehmen
- vor `push` alle Tests laufen lassen (CI + statische Codeanalyse)
- semantisch zusammenhängende Änderungen in einen Commit packen
- Feature-Branches einheitlich benennen

Es git viele weitere Befehle!

TECHNISCHE UNIVERSITÄT DARMSTADT

`git pull`	Änderungen herunterladen
`git push`	Änderungen hochladen
`git add <-A>`	Dateien hinzufügen
`git commit <-a>`	Commit erstellen
`git diff`	Versionsunterschiede anzeigen
`git log`	Versionsgeschichte anzeigen
`git show`	Details eines Commits anzeigen
`git merge`	Branches zusammenführen
`git help`	Hilfe zu Git
`git <Befehl> --help`	Hilfe zu Git-Befehlen

⇒ **Siehe Cheat-Sheet**

Vielen Dank für eure Aufmerksamkeit!

TECHNISCHE UNIVERSITÄT DARMSTADT

- Material auf d120.de/git
- Übungen
- Folien
- Cheat-Sheet mit den wichtigsten Befehlen

https://d120.de/workshops

Vielen Dank für eure Aufmerksamkeit!

TECHNISCHE UNIVERSITÄT DARMSTADT

Abbildung: https://xkcd.com/1597/

Weitere Informationen

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausblick

- Rebasing (Versionsgeschichte umschreiben)
- uvm.

Weitere Materialien

- Offizielle Dokumentation: https://git-scm.com/doc
- Interaktives Tutorial von GitHub: https://try.github.io/
- *Pro Git* Buch: https://git-scm.com/book/de