

Einfaches Branching und Merging

Ein Stärke von Git ist es, dass die Entwicklung in parallelen Zweigen stattfinden kann. Wir betrachten das hier erst mal nur für einen einfachen Fall, um dafür ein Gefühl zu bekommen.

Wir erstellen ein Bluej Projekt, das wir mit git verwalten wollen. Dabei verwenden wir zunächst nicht die eingebaute git Funktionalität von Bluej, hier soll es darum gehen, wie man mit einfachen Verzweigungen (Branches) arbeiten kann. Du kannst das Beispiel einfach nachvollziehen, wir verwenden einfach die Einstiegsübungen von dieser [Wiki-Seite](#).

- Lege ein neues Bluej-Projekt an.
- Öffne eine Kommandozeile im Projektverzeichnis
- Initialisiere ein git-Repository im Projektverzeichnis.

Lösungsvorschläge zu den Programmieraufgaben findest du auf [dieser Seite](#), von dort kannst du Lösungen für einzelne Aufgaben übernehmen, ohne alles jetzt programmieren zu müssen. So kannst du dich auf das Arbeiten mit git konzentrieren.



(A1)

Übernehme den "langen" Lösungsvorschlag für die Modulo-Methode in dein Projekt. Teste die Funktionalität und erstelle einen ersten Commit.

The screenshot shows the BlueJ IDE interface. On the left, there's a sidebar with buttons for 'Neue Klasse...', 'Übersetzen', 'Teamwork', and 'Testing'. The main window displays a Java class named 'Uebungen'. The code is as follows:

```
1 /**
2  * Beschreiben Sie hier die Klasse Uebungen.
3  *
4  *
5  * @author (Ihr Name)
6  * @version (eine Versionsnummer oder ein Datum)
7  */
8 public class Uebungen
9 {
10
11     /**
12      * Konstruktor für Objekte der Klasse Uebungen
13      */
14     public Uebungen()
15     {
16     }
17
18     public int MyModulo(int a, int b)
19     {
20         int remainder=0;
21         // dein Code
22         remainder = a % a/b*b;
23         // Rückgabe
24         return remainder;
25     }
26 }
```

A red text overlay on the right side of the code editor reads: **Lösung A1 reinkopieren, testen, Commit erstellen.**

At the bottom left of the IDE, there is a button labeled 'A02' and a link 'Lösungsvorschlag A02'. The status bar at the bottom indicates 'Klasse übersetzt - keine Syntaxfehler'.

Ändere die Modulo-Methode jetzt so ab, dass sie dem zweiten, sehr kurzen Lösungsvorschlag entspricht. Erstelle einen weiteren Commit mit sinnvoller Commit Message.

Im Ergebnis könnte das dann so aussehen:

```
$ git lg
* b664f26 - (HEAD -> main) Modulo Methode gekürzt (vor 13 Sekunden)
* 143f0be - Erster commit (vor 6 Minuten)
```

"Löse" die Aufgabe 2, verfare dabei analog zur Modulo Methode: Übernehme zunächst den ersten Lösungsvorschlag, erstelle einen Commit, ändere die Methode dann so, dass sie dem zweiten Lösungsvorschlag entspricht und erstelle dann einen weiteren Commit.

Als Zwischenergebnis erhält man einen Verlauf wie den Folgenden:

```
$ git lg
* 118e70a - (HEAD -> main) Tauschen Methode wie in Vorschlag 2 (vor 3 Sekunden)
* 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 25 Minuten)
* b664f26 - Modulo Methode gekürzt (vor 30 Minuten)
* 143f0be - Erster commit (vor 35 Minuten)
```

Einen weiteren Zweig (Branch) erstellen

Erstelle mit dem Befehl `git branch development` einen "Entwicklungszweig", in dem du neue Dinge testen kannst, ohne den Code in `main` zu verändern, wechsele dann auf den neuen Branch:

```
$ git branch development
$ git checkout development
Zu Branch 'development' gewechselt
$ git branch
* development
  main
```

Der Branch mit dem Sternchen ist der aktuelle Zweig.



(A2)

Füge eine Lösung für die Aufgabe 3 am Ende deiner Klassendatei im Projekt ein und erstelle einen Commit. Betrachte dann den Verlauf deiner Commits.

```
$ git lg --all
```

```
* aba7dbd - (HEAD -> developement) Pyramide (vor 11 Sekunden)
* 118e70a - (main) Tauschen Methode wie in Vorschlag 2 (vor 22 Minuten)
* 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 47 Minuten)
* b664f26 - Modulo Methode gekürzt (vor 51 Minuten)
* 143f0be - Erster commit (vor 57 Minuten)
```

Auf den ersten Blick ist keine "Verzweigung" erkennbar - woran kann man an der Ausgabe dennoch erkennen, dass es eine Verzweigung im Verlauf der Commits gibt?

Erklärung

HEAD zeigt auf developement, main ist im Verlauf "zurückgeblieben".

Ein neue Anforderung macht es jetzt nötig, dass im Hauptentwicklungszweig (main) sofort die Lösung für Aufgabe 4 eingefügt wird.

Gehe dazu wie folgt vor:

- Stelle sicher, dass dein Arbeitsverzeichnis "clean" ist, also keine Änderungen hat, dich noch nicht committed sind.
- Wechsle zum main Branch: `git checkout main`
- Öffne in BlueJ deine Klassendate neu - du erkennst, dass die Lösung für das Pramidenvolumen jetzt wieder verschwunden ist.
- Füge den Lösungsvorschlag für Aufgabe 4 **direkt nach dem Konstruktor**¹⁾ in deine Klasse ein und erstelle einen Commit.
- Betrachte den Verlauf der Commits mit `git lg --all`

Jetzt ist die Verzweigung direkt erkennbar:

```
$ git lg --all
* 9b12ec6 - (HEAD -> main) Alterstest (vor 3 Sekunden)
| * aba7dbd - (developement) Pyramide (vor 16 Minuten)
|/
* 118e70a - Tauschen Methode wie in Vorschlag 2 (vor 37 Minuten)
* 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 62 Minuten)
* b664f26 - Modulo Methode gekürzt (vor 67 Minuten)
* 143f0be - Erster commit (vor 72 Minuten)
```



(A3)

- Wechsle wieder zum developement Branch
- Füge die Lösungen für die Aufgaben 5 und 6 am Ende deiner Klasse ein
- Erstelle nach jeder neuen Lösung einen Commit

Deine entstehende Commit-History sollte in etwas so aussehen:

```
$ git lg --all
* 74b05e3 - (HEAD -> developement) schulnoten (vor 13 Sekunden)
* 4370307 - Geradengleichung (vor 73 Sekunden)
* aba7dbd - Pyramide (vor 20 Minuten)
| * 9b12ec6 - (main) Alterstest (vor 5 Minuten)
|/
* 118e70a - Tauschen Methode wie in Vorschlag 2 (vor 42 Minuten)
* 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 67 Minuten)
* b664f26 - Modulo Methode gekürzt (vor 71 Minuten)
* 143f0be - Erster commit (vor 77 Minuten)
```

Merging

Deine Entwicklungsarbeiten sind nun abgeschlossen, du möchtest die Änderungen beider Zweige wieder zusammenführen (mergen).

Das prinzipielle Vorgehen dabei ist folgendes:

- Wechsle auf den Branch in den du die Änderungen zusammenführen möchtest
- Führe dort den Befehl `git merge <BRANCHNAME>` aus, wobei `<BRANCHNAME>` der Branch ist, den du in den aktuellen Branch einfließen lassen willst.
- Dabei entsteht ein neuer Commit, für den du eine Commit Message eingeben musst.

Führe den Merge von `developement` in `main` durch. Wenn du dich an die Anweisungen oben gehalten hast, sollte das ohne Konflikte automatisch gelingen. Betrachte die Klassendatei, die du nach dem Merge vorfindest.

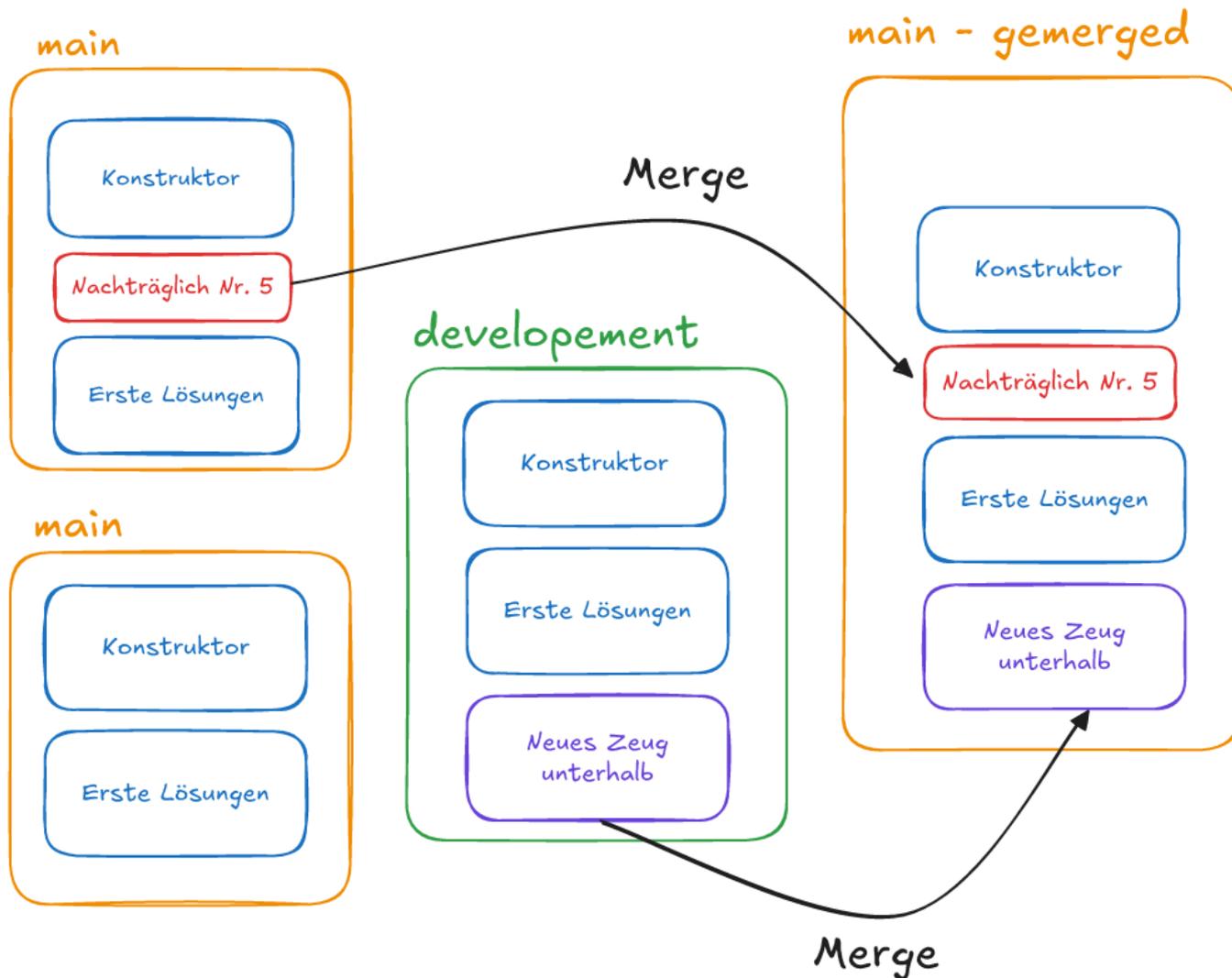
```
$ git lg --all
* 338bca1 - (HEAD -> main) Merge branch 'developement' (vor 55 Sekunden)
|\
| * 74b05e3 - (developement) schulnoten (vor 4 Minuten)
| * 4370307 - Geradengleichung (vor 5 Minuten)
| * aba7dbd - Pyramide (vor 24 Minuten)
* | 9b12ec6 - Alterstest (vor 8 Minuten)
|/
* 118e70a - Tauschen Methode wie in Vorschlag 2 (vor 45 Minuten)
* 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 70 Minuten)
* b664f26 - Modulo Methode gekürzt (vor 75 Minuten)
* 143f0be - Erster commit (vor 80 Minuten)
```

Der Branch `developement` zeigt jetzt immer noch auf den letzten Commit des Entwicklungszweigs, den benötigen wir jetzt nicht mehr, wir können ihn löschen - die Commits bleiben dabei erhalten:

```
$ git branch -d developement
Branch developement entfernt (war 74b05e3).
```

Konflikte

Git hat beim zusammenführen der beiden Zweige automatisch einen Merge gemacht, weil wir die Lösung für Aufgabe 5 direkt nach dem Konstruktor eingefügt hatten und nicht nachträglich ans Ende des Main Zweiges:



Es gab also keine Zeile in der Datei, die ausgehend vom letzten gemeinsamen Commit "doppelt geändert" war, somit konnte git die Datei automatisch zusammenführen. Was git niemals machen kann, auch wenn es keine Konflikte feststellt, ist, das entstehende Programm auf Sinnhaftigkeit zu prüfen. In unserem Fall steht jede Methode für sich und das Ergebnis macht Sinn, dsas muss aber nicht so sein. Ein konfliktfreier Merge kann zu Code führen der fehlerhaft ist, z.B. weil es eine Methode mit identischer Signatur doppelt gibt.



(A4)

- Erstelle einen neuen Zweig `development`
- Wechsle nach `development`
- Füge ans Ende die Lösung für Aufgabe 8 ein
- Erstelle einen Commit
- Wechsle zurück zu `main`
- Füge dort ans Ende der Klasse die Lösung für Aufgabe 9 ein
- Erstelle einen Commit

Das Zwischenergebnis sollte etwa so aussehen:

```
$ git lg --all
* 9c07a5c - (HEAD -> main) Dual nach Dezimal (vor 10 Sekunden)
| * 431bf5a - (development) Stellenzaehler (vor 66 Sekunden)
|/
* 338bca1 - Merge branch 'development' (vor 30 Minuten)
|\
| * 74b05e3 - schulnoten (vor 32 Minuten)
| * 4370307 - Geradengleichung (vor 33 Minuten)
| * aba7dbd - Pyramide (vor 52 Minuten)
* | 9b12ec6 - Alterstest (vor 37 Minuten)
|/
* 118e70a - Tauschen Methode wie in Vorschlag 2 (vor 74 Minuten)
* 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 2 Stunden)
* b664f26 - Modulo Methode gekürzt (vor 2 Stunden)
* 143f0be - Erster commit (vor 2 Stunden)
```

Wechsle jetzt nach `main` und führe `development` dorthin zusammen - was passiert? Wie stellt sich das im Editor von BlueJ dar?

[Hint](#)

```
$ git merge development
automatischer Merge von Uebungen.java
KONFLIKT (Inhalt): Merge-Konflikt in Uebungen.java
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
```

```

94     return textnote;
95 }
96 }
97
98 <<<<<<< HEAD
99 public int Dual2dec(int dual)
100 {
101     int stelligenziffer=0;
102     int stelligenwert=1;
103     int dezimalwert=0;
104
105     while (dual > 0) {
106         stelligenziffer = dual % 10;
107         dezimalwert = dezimalwert + stelligenziffer * stelligenwert;
108         stelligenwert = stelligenwert * 2;
109         dual = dual / 10;
110     }
111
112     return dezimalwert;
113 }
114
115 =====
116 public int Stellenzaehler(int zahl)
117 {
118     int stelligen=0;
119
120     while (zahl > 0) {
121         zahl = zahl / 10;
122         stelligen++;
123     }
124
125     return stelligen;
126 >>>>>> developement
127 }
128
129

```

Git hat jetzt "Konfliktmarker" eingefügt, weil wir in beiden Branches überschneidende Bereiche der Datei geändert haben. Wir müssen nun händisch alle Bereiche zwischen den Konfliktmarker so bearbeiten, wie es am Ende sein soll und die Marker entfernen. Anschließend erstellen wir einen Merge-Konflikt von Hand.

Die Konfliktlösung ist in unserem Fall einfach: Wir können beide Methoden einfach stehen lassen und die Marker entfernen.

```

$ git lg --all
* 60f0fa3 - (HEAD -> main) Merge: Konflikte von Hand entfernen (vor 4
Sekunden)
|\
| * 431bf5a - (developement) Stellenzaehler (vor 10 Minuten)
* | 9c07a5c - Dual nach Dezimal (vor 10 Minuten)
|/
* 338bca1 - Merge branch 'developement' (vor 39 Minuten)
|\
| * 74b05e3 - schulnoten (vor 42 Minuten)
| * 4370307 - Geradengleichung (vor 43 Minuten)
| * aba7dbd - Pyramide (vor 62 Minuten)
* | 9b12ec6 - Alterstest (vor 46 Minuten)

```

|/

- * 118e70a - Tauschen Methode wie in Vorschlag 2 (vor 83 Minuten)
- * 99ba536 - Tauschen Methode wie in Vorschlag 1 (vor 2 Stunden)
- * b664f26 - Modulo Methode gekürzt (vor 2 Stunden)
- * 143f0be - Erster commit (vor 2 Stunden)

1)

Nicht am Ende

From:
<https://www.info-bw.de/> -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:git:branching_merging:start

Last update: **19.03.2025 10:12**

