Wie kann man git beim Programmieren verwenden?

Bislang haben wir zu Demo-Zwecken in unserem Tagebuch geschrieben - das hatte vor allem den Vorteil, dass wir die Zeitleiste vor unserem inneren Auge hatten. Nun wollen wir uns überlegen, wie man git beim Programmieren einsetzen kann.

Dazu stellen wir uns vor, dass wir mit dem Arduino ein Auto steuern wollen, geplant sind die folgenden Projektschritte:

1. Unterprogramme wie Vorwärts, Kurven und Ähnliches, um das Auto zu steuern
2. Einbau eines Ultraschallsensors, um das Auto vor einem Hindernis zu stoppen
3. Einbau eines IR-Sensors, auslesen der IR Werte
4. Verwendung des IR-Sensors, um mit dem Auto einer schwarzen Linie auf dem Boden zu folgen

Jedes Feature ein Commit!

- Es empfiehlt sich, mindestens für jedes Feature einen eigenen Commit anzulegen.
- Wenn man ein funktionierendes Programm hat, und Änderungen vornehmen möcht, bei denen man sich unsicher fühlt, ob die geplante Programmierung klappt, kann man sich den aktuellen Stand als Commit sichern.
- Vorsicht ist bei "Backup-Commits" geboten, z.B. immer am Ende einer Unterrichtsstunde, denn in diesem Fall läuft man Gefahr, Code in einem Snapshot zu sichern, der nicht funktioniert - das ist natürlich suboptimal. Das sollte man nur mit gutem Grund tun, beispielsweise wenn man den Code auf einen externen Server "pushen" möchte, um zuhause weiter zu arbeiten - dazu später mehr.

Übung

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)

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

Ändere die Modulol-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, verfahre 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 developement` einen "Entwicklungszweig", in dem du neue Dinge testen kannst, ohne den Code in `main` zu verändern, wechsle dann auf den neuen Branch:

```
$ git branch developement
$ git checkout developement
Zu Branch 'developement' gewechselt
$ git branch
* developement
  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 commited 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 möchstest 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)
```

1)

Nicht am Ende

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

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:git:programmieren:start?rev=1742328007

Last update: **18.03.2025 20:00**