

Praktische Informatik 1

Versionsverwaltung mit git


Thomas Röfer

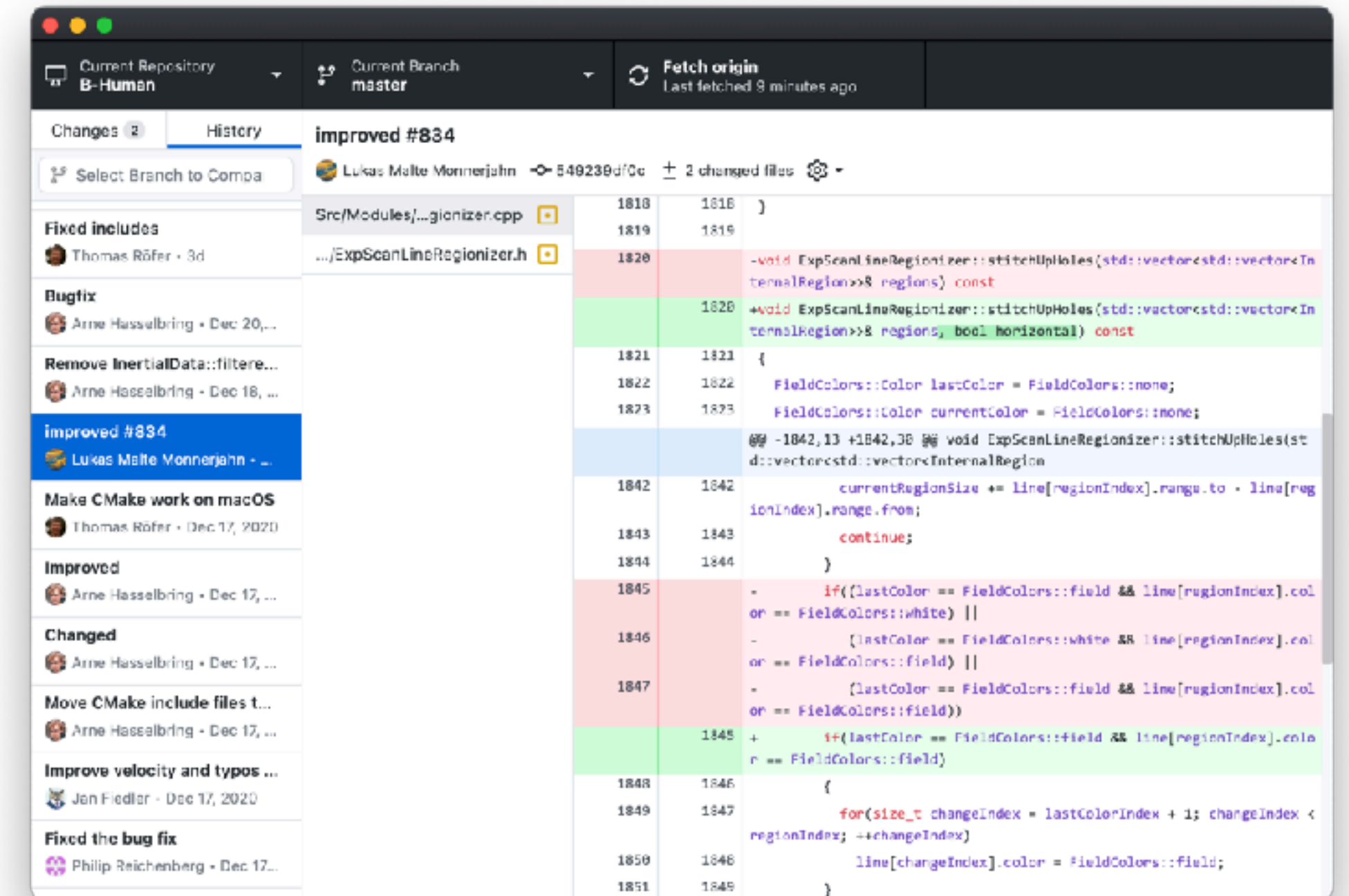
Cyber-Physical Systems
Deutsches Forschungszentrum für
Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen



Versionsverwaltung mit git

- Änderungshistorie verwalten
 - Wer hat was wann wie verändert?
 - Vergisst nie etwas, d.h. alle alten Versionen der Dateien sind noch da!
- Entwicklung wurde von Linus Torwalds initiiert
- Zusammenarbeit über das  GitLab der Informatik
 - **gitlab.informatik.uni-bremen.de** (Nicht **gitlab.com**!)
- Alle Abgaben passieren ab nun per GitLab

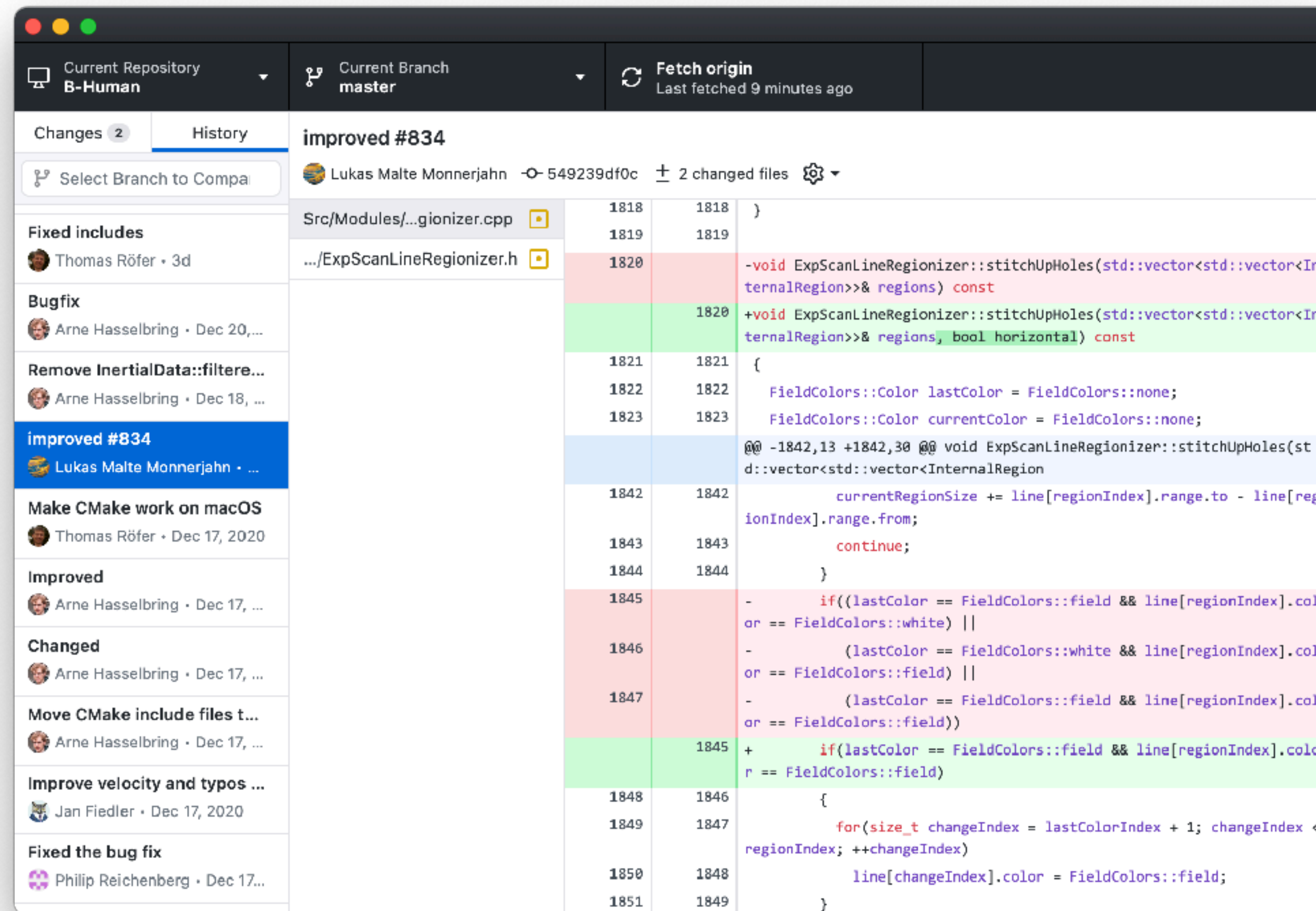


The screenshot shows a Git GUI interface. On the left, there's a sidebar with a list of commits. The selected commit is 'improved #834' by Lukas Malte Monnerjahn. The main area displays a diff for this commit, showing changes in the file 'ExpScanLineRegionizer.h'. The diff highlights several lines of code that have been modified, including a change in the 'stitchUpHoles' function and the addition of a new line for 'currentRegionSize'.

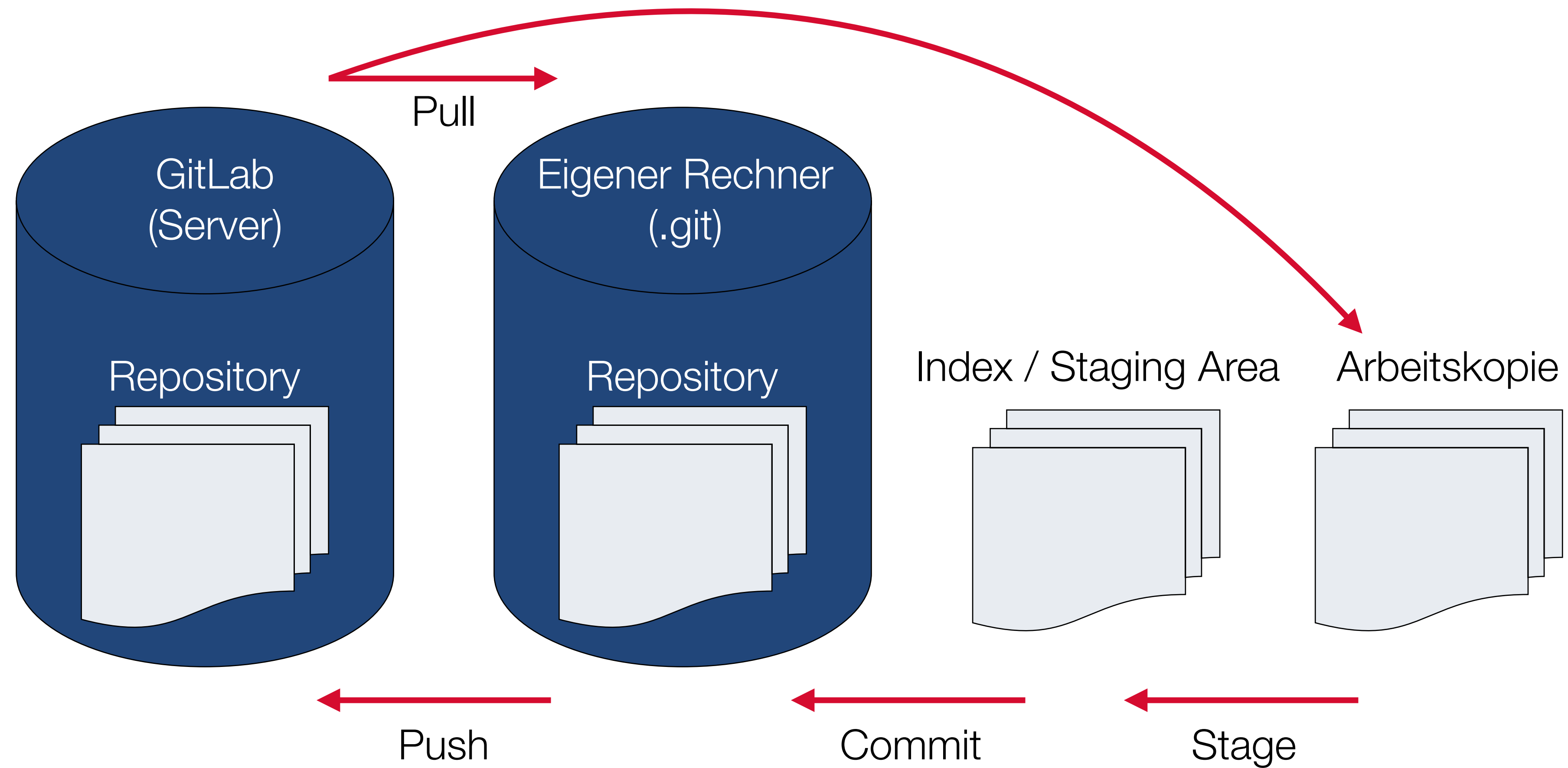


Von Krd (Foto), Sprat (Zuschnitt), CC BY-SA 4.0, Wikipedia

Entwickeln mit GitHub Desktop und GitLab: Demo

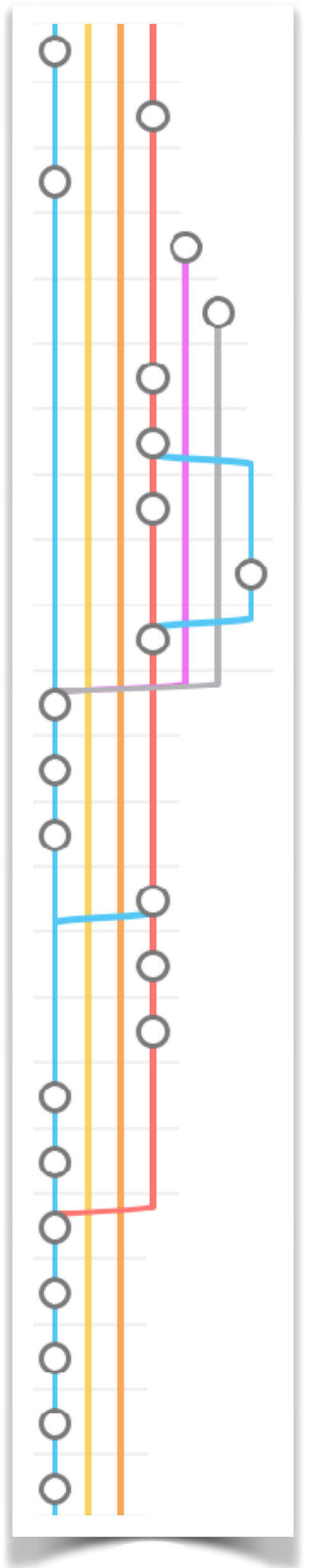


Überblick









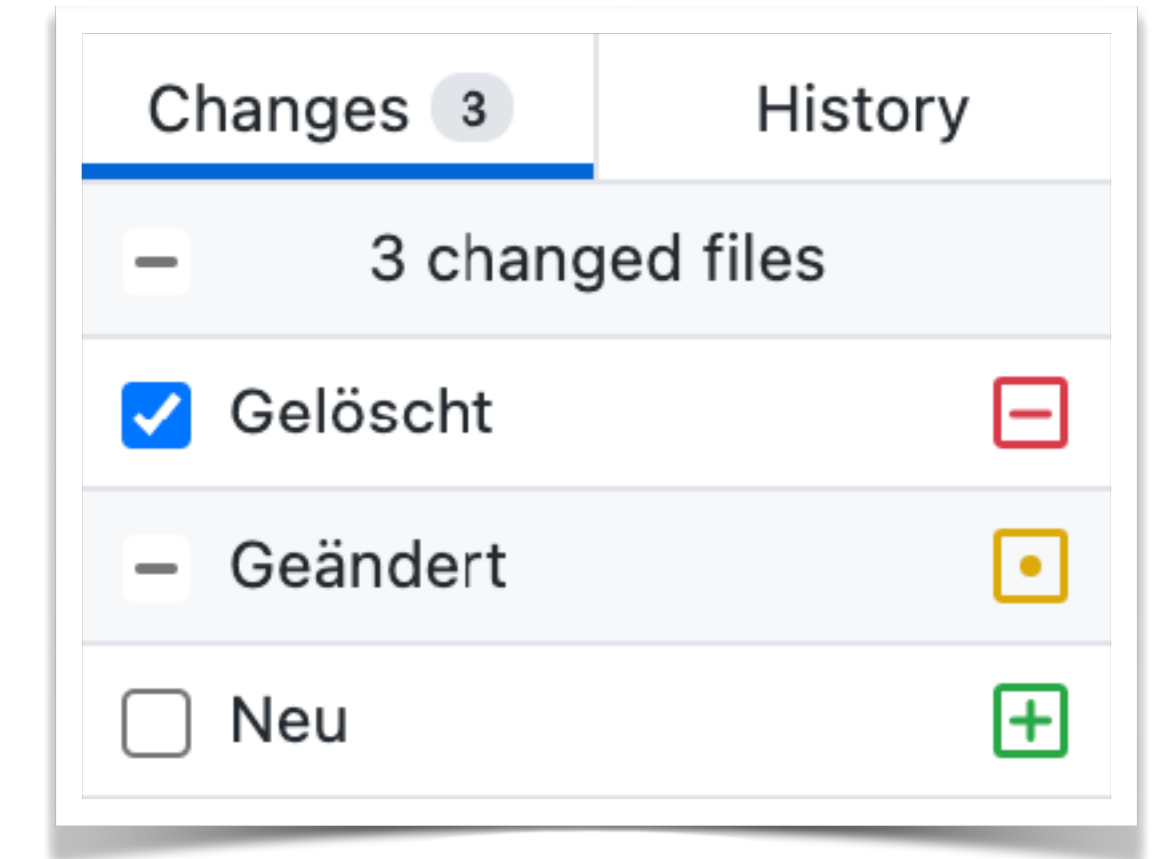
Begriffe

- **Arbeitskopie**: Dateien, an denen tatsächlich gearbeitet wird
- **Commit**: Zustand aller unter Versionsverwaltung stehender Dateien zu einem bestimmten Zeitpunkt, der explizit festgehalten wurde
 - Jeder Commit kennt seine Vorgänger-Commits → Historie
- **Branch**: Ein benannter Commit
 - Wir nutzen nur **main** und **origin/main**
- **Repository**: Eine Ansammlung von Commits mit der Markierung, welcher lokale Branch der aktuelle ist (**HEAD**)
- **Index (Staging Area)**: Explizit zusammengestellter Zustand von Dateien, aus dem der nächste Commit erzeugt werden kann



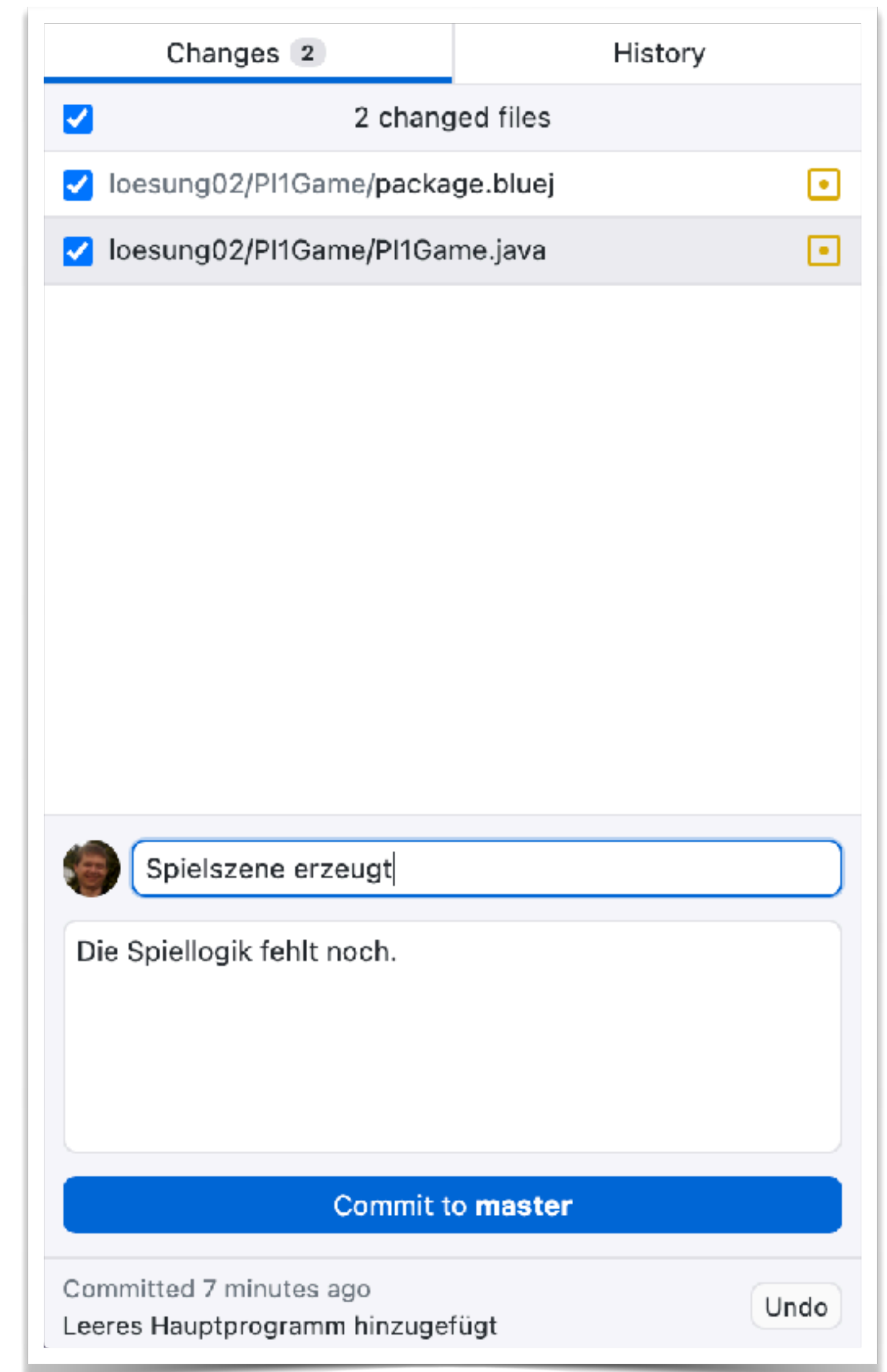
Commit vorbereiten: Staging

- Kartenreiter **Changes**
 -    Dateien sind neu, geändert oder gelöscht
 -    Änderungen werden (nicht/teilweise) Teil des nächsten Commits (**git add <Datei>** / **git reset <Datei>**)
- Einzelne Abschnitte von Textdateien lassen sich in der Ansicht rechts durch Anklicken der Zeilennummern markieren (Achtung: Für **geänderte** Zeilen muss Löschung **und** Einfügung markiert werden)
- Kontextmenü der Dateien
 - **Discard Changes...**: Datei wieder auf Zustand aus letztem Commit zurücksetzen (**git checkout -- <Datei>**)
 - **Ignore File**: Datei soll nicht unter Versionsverwaltung stehen (oft: generierte Dateien)
 - **Ignore All .<Endung> Files**: Alle Dateien mit dieser Endung ignorieren
- Ignorieren erzeugt/erweitert Datei **.gitignore** im Hauptverzeichnis, die auch Teil eines Commits werden sollte



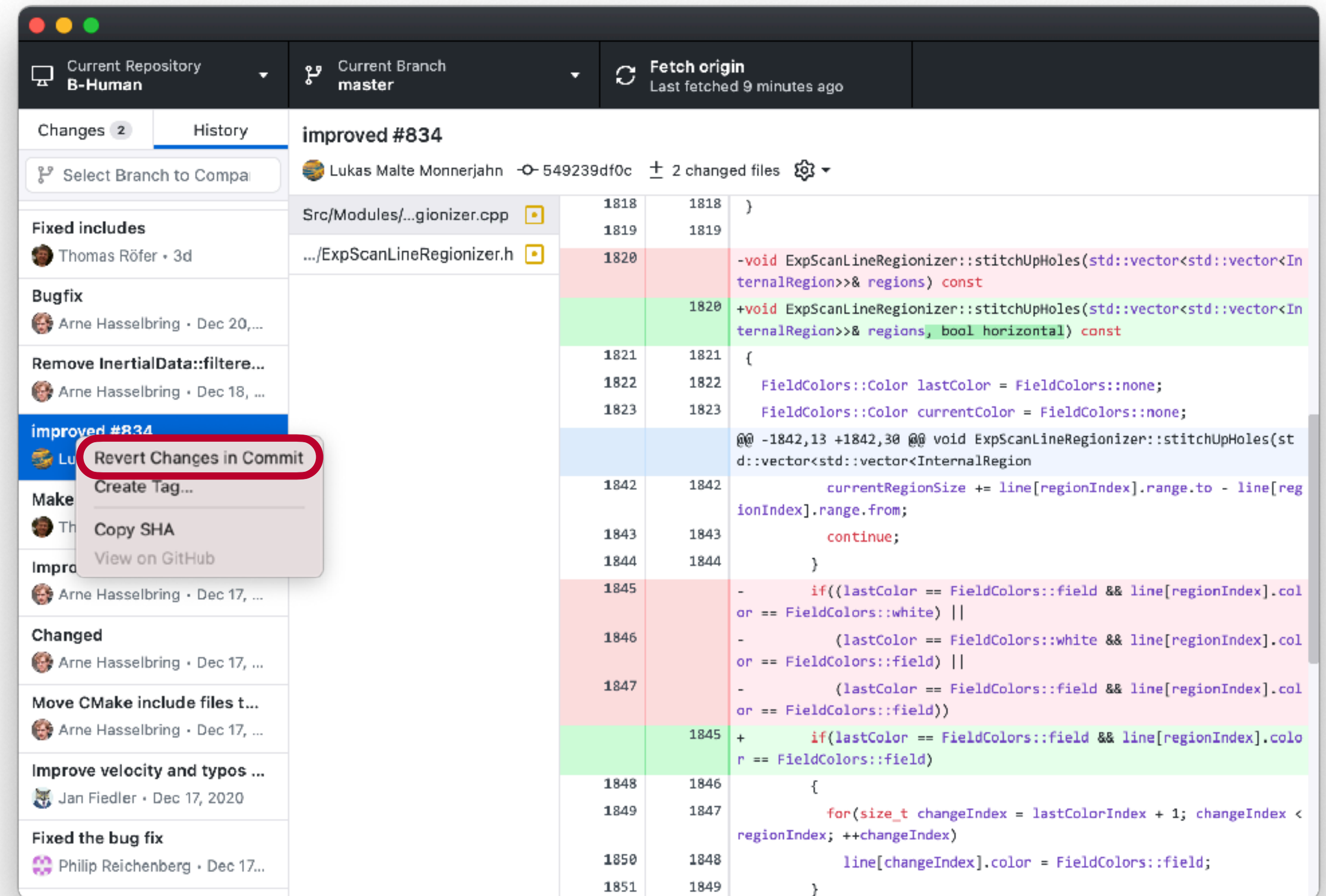
Commit (`git commit -m "<beschreibung>"`)

- Commit-Nachricht
 - Kurzbeschreibung für Bedeutung des Commits in einer Zeile (**ist ein Muss!**)
 - Zusätzliche/ausführlichere Informationen in weiterem Eingabefeld
- Commit landet erst einmal nur im Repository auf dem eigenen Rechner, **nicht auf dem Server!**
- **Undo** entfernt letzten Commit, wenn er noch nicht auf den Server übertragen wurde
 - Die Änderungen bleiben in der Arbeitskopie erhalten




Historie (git log)

- Änderungsverlauf des Repositories
 - Beschreibung der Änderung
 - Wer hat geändert (mit Bild von **gravatar.com**)?
 - Was wurde geändert?
- **Revert**: Einen Commit rückgängig machen (Erzeugt neuen „Anti“-Commit)
 - (**git revert <commit>**)
 - Kann **Konflikte** erzeugen!



Änderungen vom Server holen

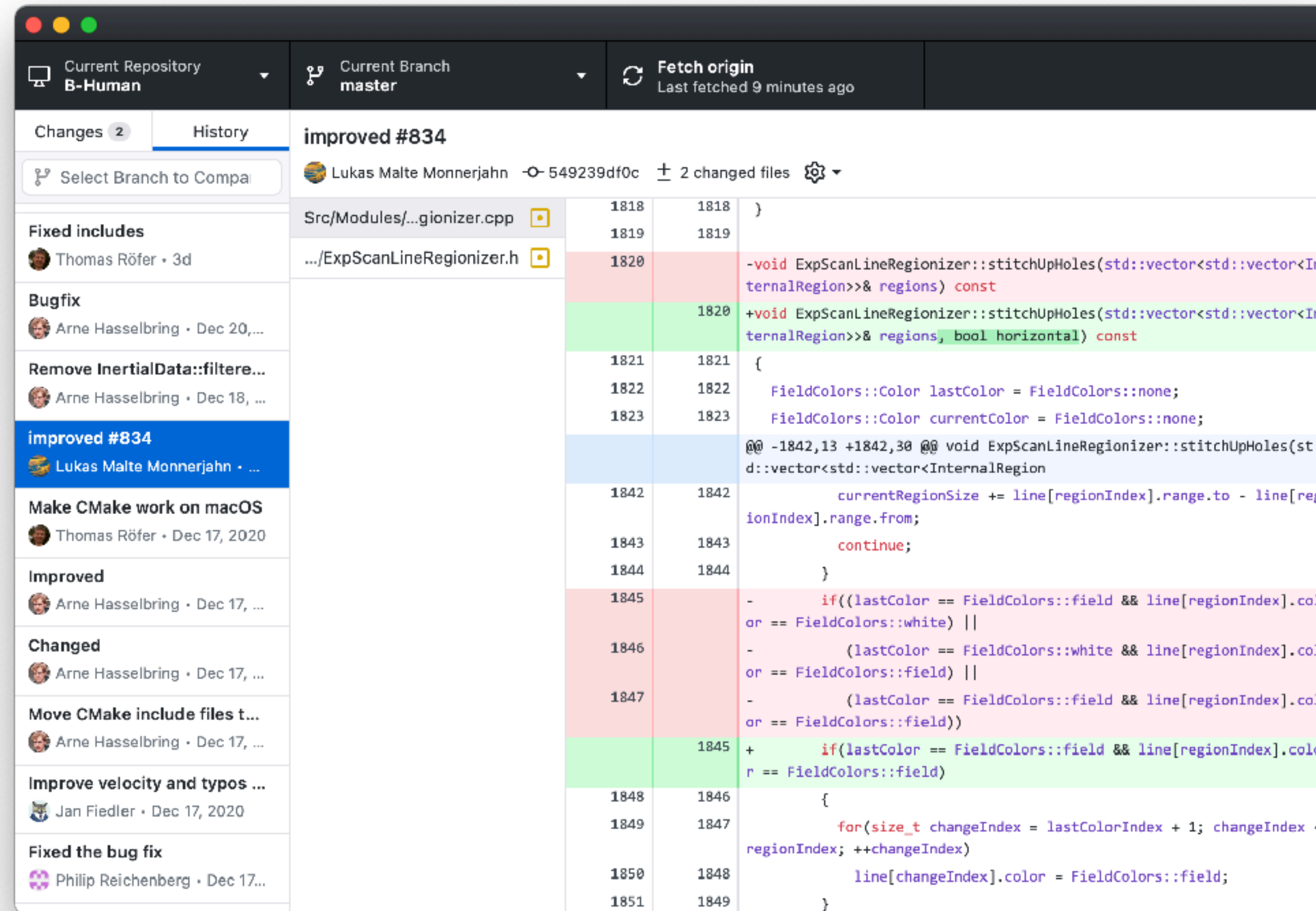
 Fetch origin
Last fetched a minute ago Pull origin
Last fetched just now

- **Fetch**: Holt neue Commits vom Server und integriert sie ins lokale Repository (**git fetch**)
- **Pull**: Wie **Fetch**, aber integriert neue Commits auch in die Arbeitskopie (**git pull**)
 - GitLab Desktop sichert vorher automatisch geänderte Dateien in der Arbeitskopie und stellt sie nachher wieder her (**git pull --autostash**)
 - **Pull** kann **Konflikte** erzeugen!
- Gibt es lokal und auf dem Server neue Commits, wird ein **Merge Commit** erstellt, der zwei Eltern-Commits hat
 - Eventuell aufgelöste Konflikte werden Teil des **Merge Commits**

Änderungen auf Server übertragen (**git push**)

- Spielt alle lokalen Commits auf den Server, die dort noch fehlen
- Geht nur, wenn alle Commits des Servers lokal vorhanden sind
- Ablauf
 - (**Fetch** und) **Pull**, um alle Commits des Servers zu holen
 - **Testen**, ob diese zusammen mit den lokalen Änderungen funktionieren (keine Fehler auf den Server spielen)!
 - **Push**, wenn inzwischen keine neuen Commits auf dem Server sind

Konflikte: Demo



Current Repository: B-Human | Current Branch: master | Fetch origin (Last fetched 9 minutes ago)

Changes: 2 | History

Select Branch to Compare

Fixed includes
Thomas Röfer • 3d

Bugfix
Arne Hasselbring • Dec 20, ...

Remove InertialData::filter...
Arne Hasselbring • Dec 18, ...

improved #834
Lukas Malte Monnerjahn • ...

Make CMake work on macOS
Thomas Röfer • Dec 17, 2020

Improved
Arne Hasselbring • Dec 17, ...

Changed
Arne Hasselbring • Dec 17, ...

Move CMake include files t...
Arne Hasselbring • Dec 17, ...

Improve velocity and typos ...
Jan Fiedler • Dec 17, 2020

Fixed the bug fix
Philip Reichenberg • Dec 17...

improved #834
Lukas Malte Monnerjahn • 549239df0c • 2 changed files

File	Line	Diff	Code
Src/Modules/...gionizer.cpp	1818	1818	}
...	1819	1819	}
.../ExpScanLineRegionizer.h	1820	1820	-void ExpScanLineRegionizer::stitchUpHoles(std::vector<std::vector<InternalRegion>>& regions) const
		1820	+void ExpScanLineRegionizer::stitchUpHoles(std::vector<std::vector<InternalRegion>>& regions, bool horizontal) const
	1821	1821	{
	1822	1822	FieldColors::Color lastColor = FieldColors::none;
	1823	1823	FieldColors::Color currentColor = FieldColors::none;
			@@ -1842,13 +1842,30 @@ void ExpScanLineRegionizer::stitchUpHoles(std::vector<std::vector<InternalRegion
	1842	1842	currentRegionSize += line[regionIndex].range.to - line[regionIndex].range.from;
	1843	1843	continue;
	1844	1844	}
	1845		- if((lastColor == FieldColors::field && line[regionIndex].color == FieldColors::white)
	1846		- (lastColor == FieldColors::white && line[regionIndex].color == FieldColors::field)
	1847		- (lastColor == FieldColors::field && line[regionIndex].color == FieldColors::field))
		1845	+ if(lastColor == FieldColors::field && line[regionIndex].color == FieldColors::field)
	1848	1846	{
	1849	1847	for(size_t changeIndex = lastColorIndex + 1; changeIndex < regionIndex; ++changeIndex)
	1850	1848	line[changeIndex].color = FieldColors::field;
	1851	1849	}

Konflikte vermeiden

- Nur Quelldateien unter Versionsverwaltung stellen
- Generierte Dateien ignorieren
 - In GitLab im Hauptverzeichnis des Repositories Datei **.gitignore** erstellen, Inhalt der **gitignore.txt** aus Stud.IP einfügen und commit-ten
- **BlueJ** vor Arbeit mit **git** beenden
 - Schreibt Projektdatei einheitlich neu
 - Laufendes **BlueJ** bemerkt Änderungen durch **Pull** vielleicht nicht
- **Discard** für **package.bluej** ausführen, wenn Klassenanordnung nicht geändert wurde

```
# Temporäre BlueJ-Dateien
*.class
*.ctxt
**/doc/*.html
**/doc/*.js
**/doc/*.zip
**/doc/element-list
**/doc/jquery
**/doc/resources
**/doc/styleSheet.css
**/doc/logfile.txt
**/doc/package-list
**/doc/legal

# Temporäre TeX-Dateien
*.aux
*.bbl
*.blg
*.fdb_latexmk
*.fls
*.log
*.synctex.gz
*.toc

# Versteckte Dateien (aber .gitignore behalten)
.*
!.gitignore
```


Konflikte auflösen

- Textdateien: Datei mit Konflikten in Texteditor öffnen
 - Text zwischen Konfliktmarkierungen geeignet kombinieren
 - Konfliktmarkierungen entfernen
- Andere Dateien: In Menü **Resolve** eigene Version (**main**) oder die vom Server (**origin/main**) auswählen
- Danach **Commit Merge** auswählen

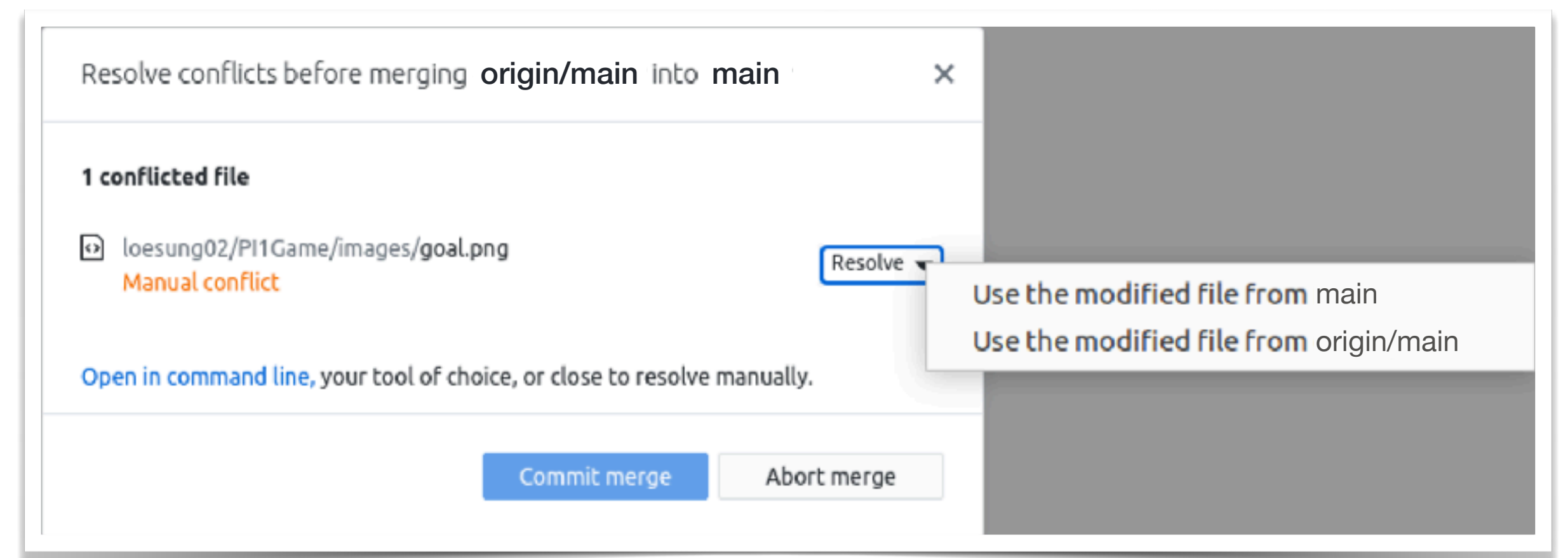
Eigene

Server

```

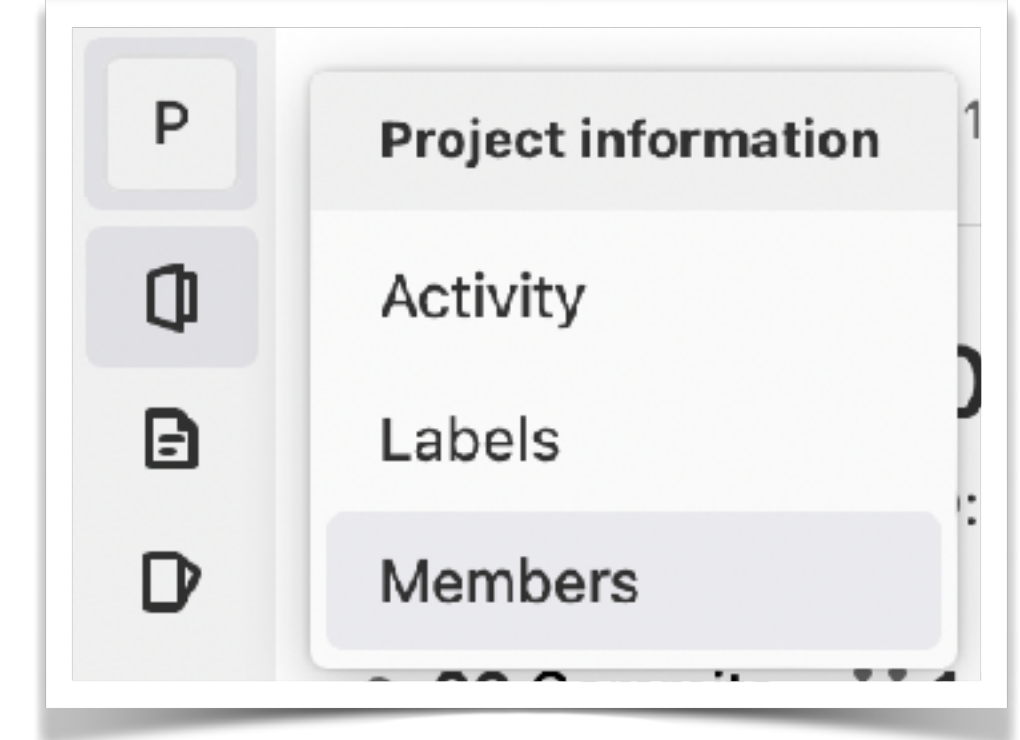
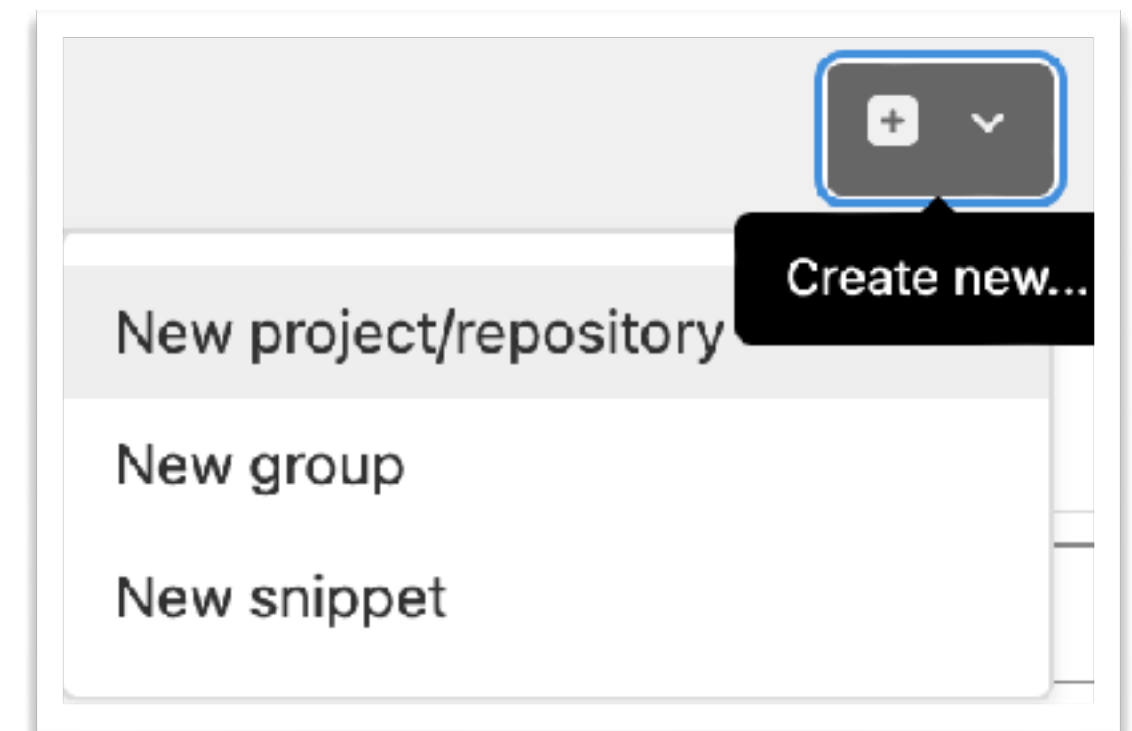
<<<<<<< HEAD
    final int key = getNextKey();
    if (key == VK_RIGHT) {
        player.setRotation(0);
        :
    else {
        playSound("error");
        continue;
    }

=====
    switch (getNextKey()) {
    case VK_RIGHT -> {
        player.setRotation(0);
        :
    default -> {
        playSound("error");
    }
  }
>>>>>>> a1d60520c29b360f1822a0bb922fd3998f9bb421
  
```

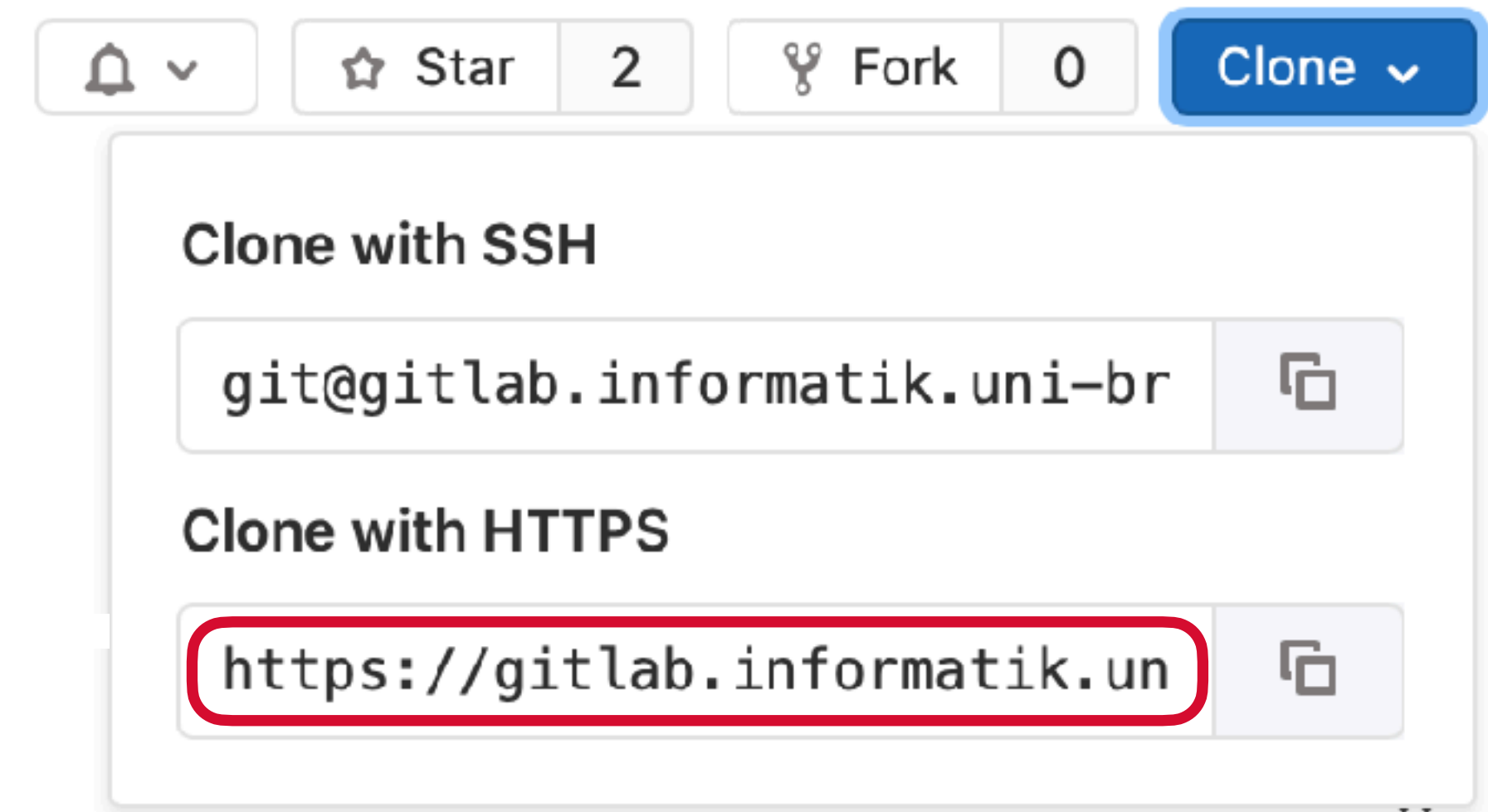
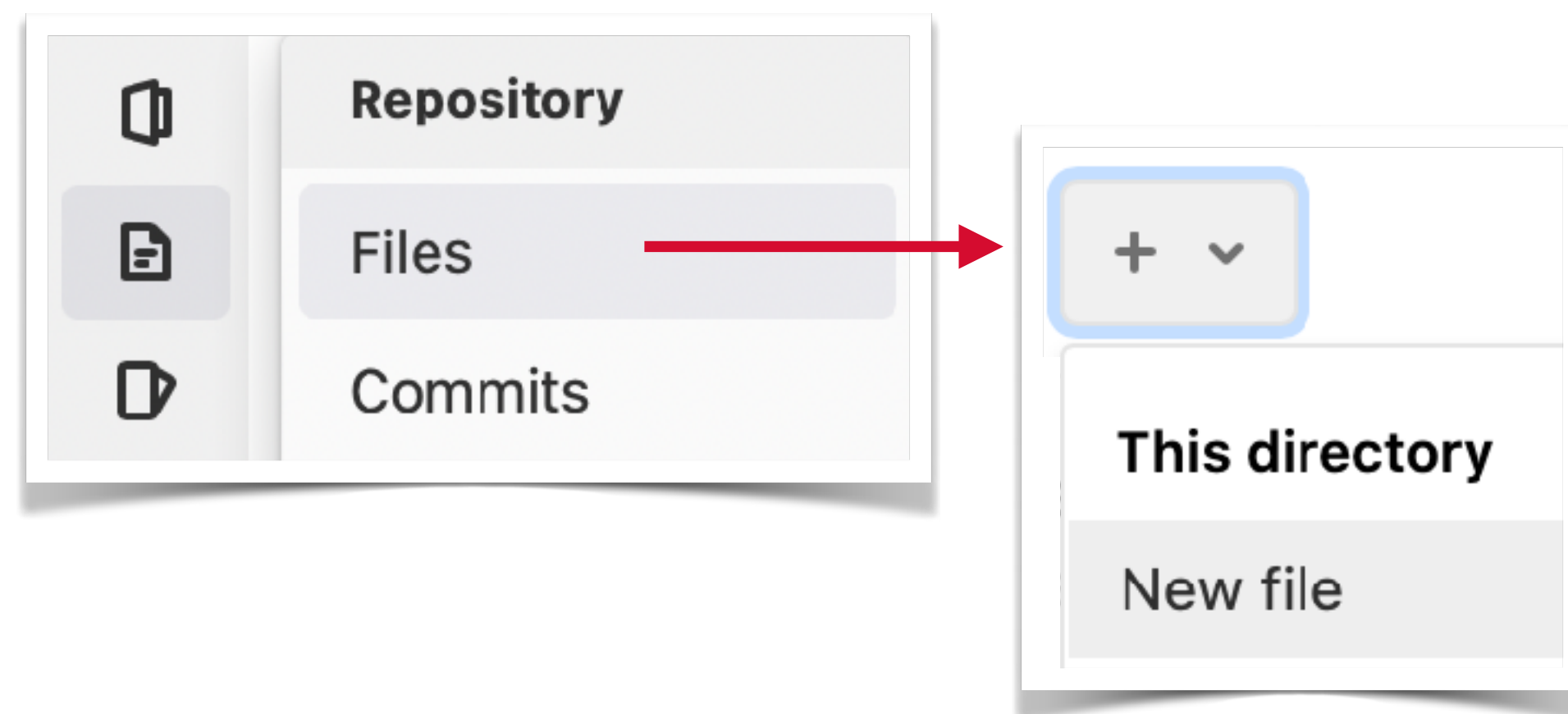


Repository einrichten (`git clone <Adresse>`)

- Neues Projekt **pi1-2022** erzeugen und eure Tutor:in als **Developer** einladen
- In GitLab die Datei **.gitignore** erstellen, Inhalt der **gitignore.txt** aus Stud.IP einfügen und commit-ten
- Repository mit Git(Hub Desktop) von **HTTPS**-Adresse klonen



oder



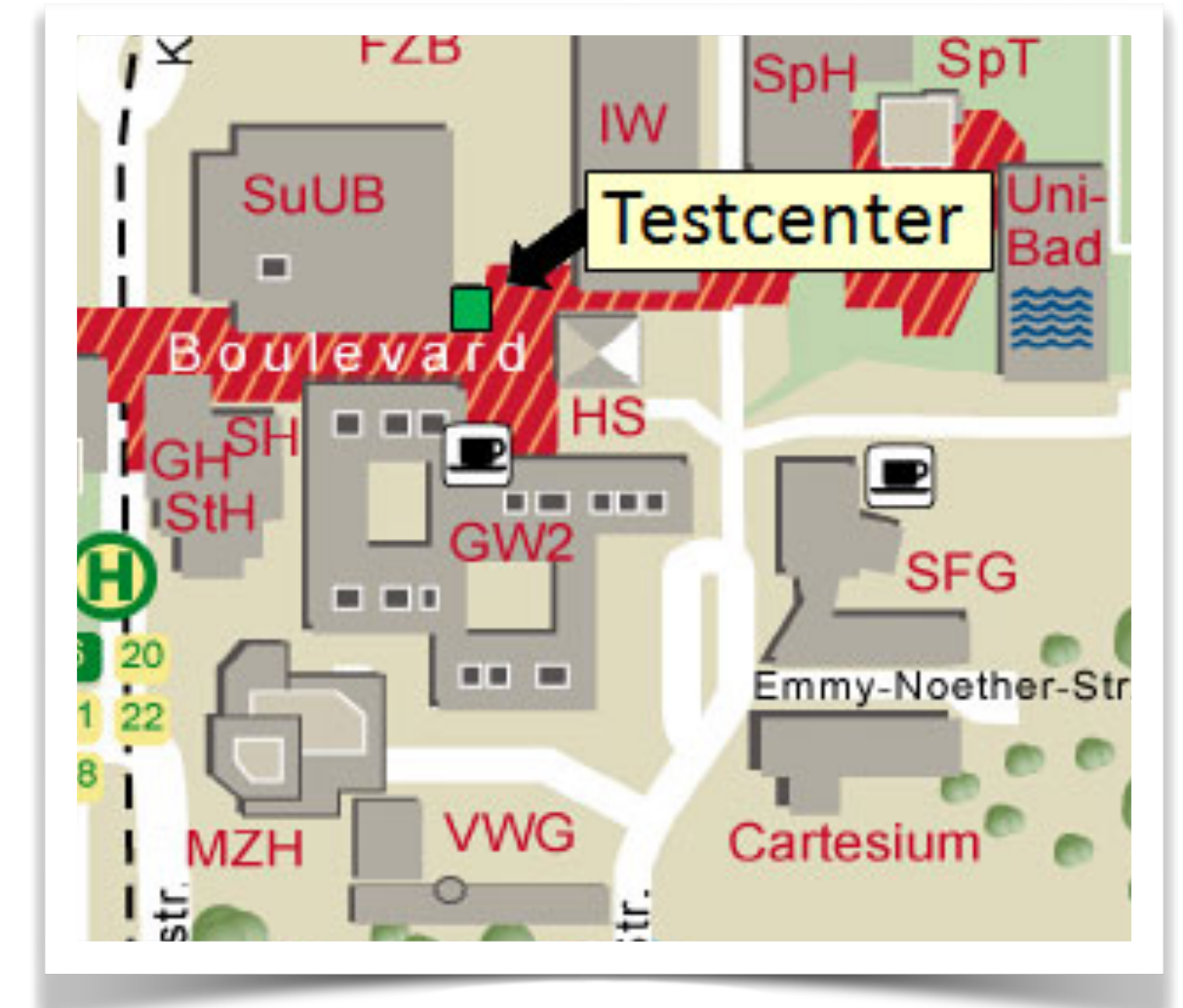
Zusammenfassung der Konzepte

- **Versionsverwaltung**
- **Arbeitskopie, Index, Repository** (lokal und auf Server)
- **Commit**
- **Pull** und **Push**
- **Konflikte** vermeiden und auflösen

Auf git's!

Übungsblatt 7 (Übungsklausur)

- 12./14.12, jeweils 10:00, 12:00, 14:00, 16:00, Einlass **15 Min.** vorher
- Einzelbearbeitung am Rechner (1¾ Stunden)
 - Programmieraufgaben: 100%
 - Fragenkatalog: 20%
- Anmeldung unter **Terminvergabe** im Stud.IP bis 04.12 (bevorzugt überlappend mit eigenem Tutoriumstermin)
 - Keine Anmeldung → keine Klausur → keine Punkte für Übungsblatt 7
- Beachtet **<https://www.uni-bremen.de/zmml/kompetenzbereiche/e-assessment/>**



Übungsklausur: Programmieraufgaben

- Einrichtung über **install.cmd** auf dem Desktop, erzeugt Projekt **Uebungsklausur2022** auf dem Desktop
- Aufgaben werden in BlueJ über **Neue Klasse...** erzeugt
 - Kommentare erhalten, damit Zuordnung zu Aufgaben bei automatischer Korrektur möglich
 - Jede Aufgabe nur einmal abgeben (Doppelte entfernen)
- Nutzen **getX()**, **getY()** und **setRotation()** aus **GameObject**
- Jede Aufgabe hat eine **main()**-Methode zum Testen

Übungsklausur: Programmieraufgaben

- **Aufgabe 1 (50P)**: Attribute, Konstruktor, Verzweigung (ähnlich zu Übungsblatt 3)
- **Aufgabe 2 (25P)**: Aufgabe 1 + Suchschleife, Array, String (ähnlich zu Übungsblatt 5)
- **Aufgabe 3 (25P)**: Aufgabe 2 + sich selbst Lösung überlegen
- **Eigener Code** kann von **Aufgabe n** zu **Aufgabe n+1** kopiert werden (d.h. vorgegebenen Code **nicht** ersetzen)
- Alles, was nicht kompiliert, gibt dabei **0 Punkte**