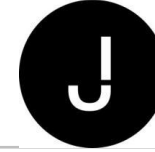




**JAVACREAM**

*Training  
Consulting  
Projectmanagement*

# GIT



- Name und Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemstellung
- Individuelle Zielsetzung

# Einführung

- Verwaltung von “Meta-Informationen” wird übernommen
  - “wer hat wann was warum gemacht”
- Bestimmte Dateien werden in bestimmten Ständen zu einem Gesamt-Stand gruppiert
- Stände können parallel existieren
- Konsolidierung von Ständen
- Tooling, Historischer Verlauf, Unterschiede in Dateien/Ständen, ...
- Gemeinsamer Zugriff durch einen Server (Authentifizierung, ...)
- Team-Zusammenarbeit

- Verwaltung von “Meta-Informationen” wird übernommen
  - “wer hat wann was warum gemacht”
- Bestimmte Dateien werden in bestimmten Ständen zu einem Gesamt-Stand gruppiert
- Stände können parallel existieren
- Konsolidierung von Ständen
- Tooling, Historischer Verlauf, Unterschiede in Dateien/Ständen, ...
  - Konsole
- Gemeinsamer Zugriff durch einen Server (Authentifizierung, ...)
- Team-Zusammenarbeit

- Verwaltung von “Meta-Informationen” wird übernommen
  - “wer hat wann was warum gemacht”
- Bestimmte Dateien werden in bestimmten Ständen zu einem Gesamt-Stand gruppiert
- Stände können parallel existieren
- Konsolidierung von Ständen
- Tooling, Historischer Verlauf, Unterschiede in Dateien/Ständen, ...
  - Konsole, Integration ins Betriebssystem, Integration in Editoren und IDEs
- Gemeinsamer Zugriff durch einen Server (Authentifizierung, ...)
- Team-Zusammenarbeit



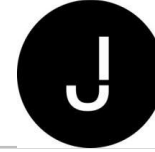
- Desktop
  - TortoiseGit
  - SourceTree
  - ...
- Plugins für Editoren
  - Eclipse
  - Visual Studio
  - Visual Studio Code
  - ...



- Verwaltung von “Meta-Informationen” wird übernommen
  - “wer hat wann was warum gemacht”
- Bestimmte Dateien werden in bestimmten Ständen zu einem Gesamt-Stand gruppiert
- Stände können parallel existieren
- Konsolidierung von Ständen
- Tooling, Historischer Verlauf, Unterschiede in Dateien/Ständen, ...
  - Konsole, Integration ins Betriebssystem, Integration in Editoren und IDEs, Web Console
- Gemeinsamer Zugriff durch einen Server (Authentifizierung, ...)
- Team-Zusammenarbeit



- GitHub
  - Server laufen in der Microsoft-Cloud
  - Öffentliche Ablage ist kostenlos, private Bereiche Lizenz-pflichtig
- BitBucket
  - Atlassian
  - Cloud-Service + Betrieb auf eigenen Servern
- GitLab
  - gitlab.com
  - Cloud-Service + Betrieb auf eigenen Servern
- Azure DevOps
  - Microsoft-Cloud

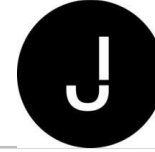


## First Contact

- Terminal-Fenster mit Git-Unterstützung
- Das Kommando “git” steht hierin zur Verfügung
  - Das ist **kein simples Command Line Interface**, das mit einem Git-Server kommuniziert
    - Es gibt keinen laufenden Git-Server-Prozess, kein Dämon, ...
  - `git --version`
  -

```
rainer@rainer-Aspire-VN7-572G:~$ git --version  
git version 2.32.0
```

- `git config <scope> <key-hierarchie> <value>`
  - `<scope>`
    - local
    - global (User-Profile)
    - system
  - `<key-hierarchie>`
    - “.” trennt die Hierarchie-Ebenen
  - `<value>`
    - irgendwas, Leerzeichen etc. aber in Anführungszeichen setzen
- `git config --global user.name “Rainer Sawitzki”`
- `git config --global user.email training@rainer-sawitzki.de`
- Auslesen `git config --get user.name`



## Erstes Arbeiten mit Git

# Einrichten eines Git-Projektverzeichnis

mkdir first  
cd first

git init

git status  
Fehlerfrei

Normales Verzeichnis -> Git-Projektverzeichnis

Git Workspace

Git Repository

Hinweis:  
In der Praxis entspricht diese Sequenz  
einem  
git clone server-repo first

```
echo Hello > readme.txt
```

Normales Verzeichnis -> Git-Projektverzeichnis

```
git add readme.txt
```

Git Workspace

readme.txt  
Hello

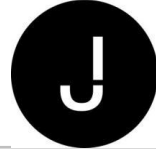
Git Repository

e9/65047ad7c57865823c7d992b1d046ea66edf78

f.....

- Was ist diese Datei  
“e9/65047ad7c57865823c7d992b1d046ea66edf78”?
  - Binärformat, das heißt Informationen sind in dieser Datei abgelegt worden
  - Der Dateiname ist
    - ~~UID, weltweit eindeutiger Zufallswert~~ oder
    - Ein berechneter Hash-Wert, berechnet aus dem Inhalt “Hello”





- e9/65047ad7c57865823c7d992b1d046ea66edf78
- Wenn in irgendeinem Git-Repository dieser Hash-Wert existiert ist es mit höchster Wahrscheinlichkeit eine Datei mit dem Inhalt “Hello”
  - Wahrscheinlichkeit einer Kollision ist absurd gering

```
echo Hello > readme.txt
```

Normales Verzeichnis -> Git-Projektverzeichnis

```
git add readme.txt
```

Git Workspace

readme.txt  
Hello

```
git commit -m "warum?"
```

Git Repository

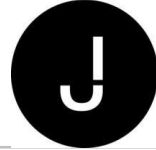
Index / Staging Area

e9/65047ad7c57865823c7d992b1d046ea66edf78

f.....

Commit  
Objekt

- Liste der Dateien / Informationen, die zu diesem Stand=Commit gehören
- Committer
  - user.name + user.email
- Timestamp
- Commit-Message
  - Angabe über Option `git commit -m "..."`
  - Ohne Option öffnet sich ein Editor-Fenster
    - vim (Drücke "i" für den Insert-Modus, Fertig: ESC, Eingabe von :wq)
    - nano
    - `git config --global core.editor notepad`



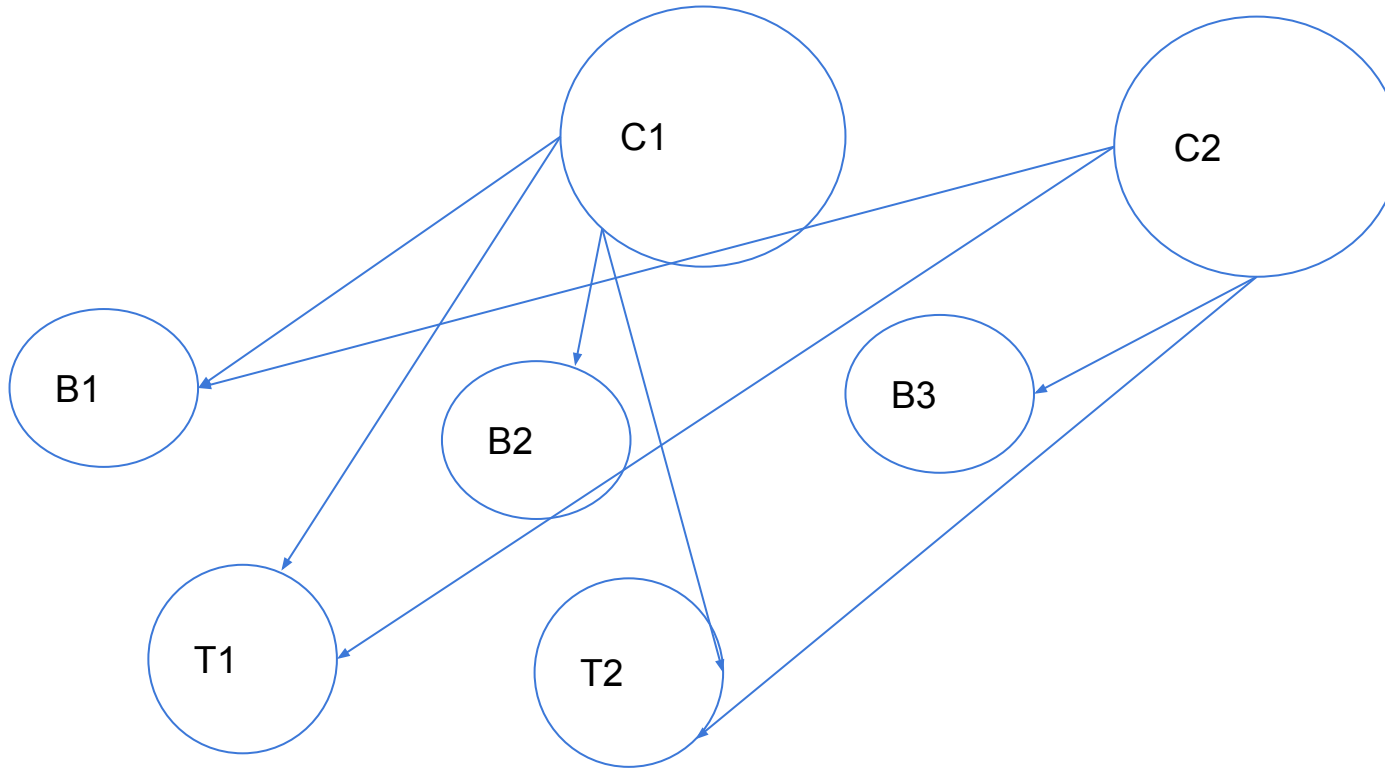
- Ist in einem Repository ein Commit mit dem Hash `ee23e95a40724fb1ad5b119d2ed9e8f7c069813e` vorhanden:
  - Rainer Sawitzki hat am um eine Commit erstellt mit der Message `add content` und den Dateien `readme.txt(Hello)` `content.txt(Hugo)`

- Allgemein
  - Dateien, die über einen Hashwert identifiziert sind
- Typen
  - Content- oder BLOB-Objekte
    - Diese repräsentieren Inhalte
  - Tree-Objekte
    - Pfad-Informationen
      - Diese werden erst beim commit erzeugt
  - Commit-Objekte
    - Diese repräsentieren einen Stand

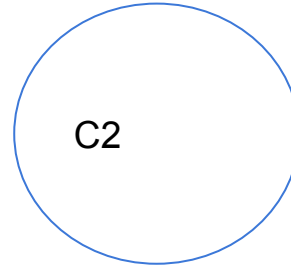
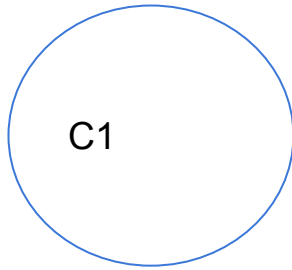
- Sie arbeiten normal im Workspace
  - Änderungen, neue Dateien, löschen
- Zweistufiger Prozess
  - Welche Dateien sollen hinzugenommen werden?
    - `git add <file> | <directory>`
      - inklusive Jokerzeichen
      - `git add .`
    - Erzeugen des Commit-Objekts
      - `git commit -m "..."`
- Vorsicht
  - "es gibt doch `git commit -a`"
  - `-a = --all`
    - `--all` bezieht sich nur auf Dateien, die schon in der Staging-Area waren



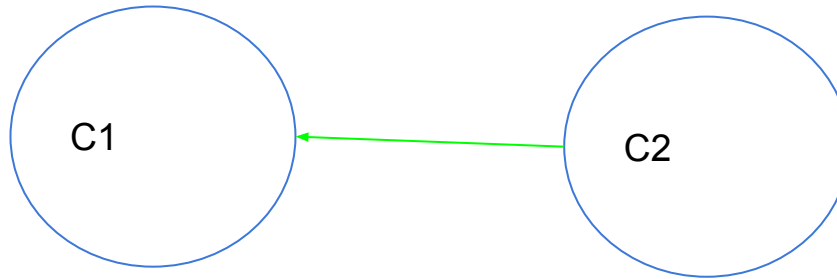
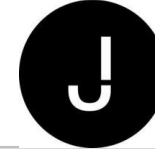
- Textdatei als Bestandteil des Workspaces
  - oder eines Unterverzeichnisses
- In dieser Textdatei werden Regeln hinterlegt, die Dateien ausschließen
  - Unterverzeichnis-.gitignores werden gemerged mit denen der Ober-Verzeichnisse
- Hinweis
  - Standard-Namen (\*.bak, ...) werden automatisch ausgeschlossen







Es werden nur  
noch die  
Commit-Objekte  
gezeichnet



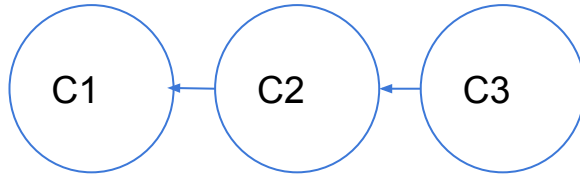


- Liste der Dateien / Informationen, die zu diesem Stand=Commit gehören
- Committer
  - user.name + user.email
- Timestamp
- Commit-Message
- Hash des Vorgänger-Commits

- Ist in einem Repository ein Commit mit dem Hash 9e7f4ac1208a5ceaa5f36e4d4a96276b9318c5fa vorhanden:
  - Rainer Sawitzki hat am um eine Commit erstellt mit der Message ... und den Dateien ... ausgehend vom Commit mit dem Hashwert .... der dann wiederum den Hash ee23e95a40724fb1ad5b119d2ed9e8f7c069813e mit Rainer Sawitzki hat am um eine Commit erstellt mit der Message add content und den Dateien readme.txt(Hello) content.txt(Hugo)
-

- Durch dieses Arbeiten mit über Hash-Werte verketteten Informationen entsteht eine unmodifizierbare, nicht nachträglich änderbare Historie von Informationen
- Grundlage dieser Technologie sind die so genannten Merkle-Trees

Fokus auf Hash-Werte, “Nerd-Modus”





- 2 Szenarien für die Einführung eines Namens
  - Definition eines fixen Standes
    - Typischerweise Versionen
      - v1.0, v1.1-Milestone1
    - “HeuteMorgen”
  - Benennung einer gerade laufenden Aktion innerhalb eines sich entwickelnden Projektes
    - Typischerweise ist das der Name eine Aktion, einer Ticket-Nummer
      - “implement\_feature1”, Jira-Ticket 0815
    - “working”, “experiment”, ...

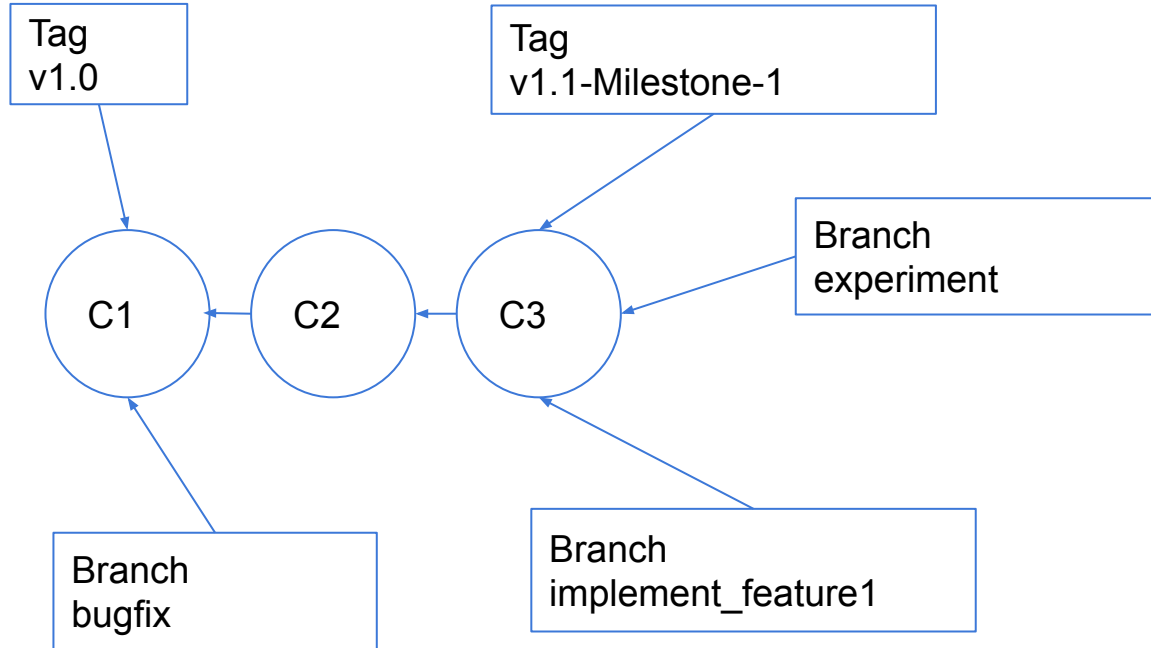


- 2 Szenarien für die Einführung eines Namens
  - Definition eines fixen Standes
    - Typischerweise Versionen
      - v1.0, v1.1-Milestone1
    - “HeuteMorgen”
    - Git Tag
  - Benennung einer gerade laufenden Aktion innerhalb eines sich entwickelnden Projektes
    - Typischerweise ist das der Name eine Aktion, einer Ticket-Nummer
      - “implement\_feature1”, Jira-Ticket 0815
    - “working”, “experiment”, ...
    - Git Branch

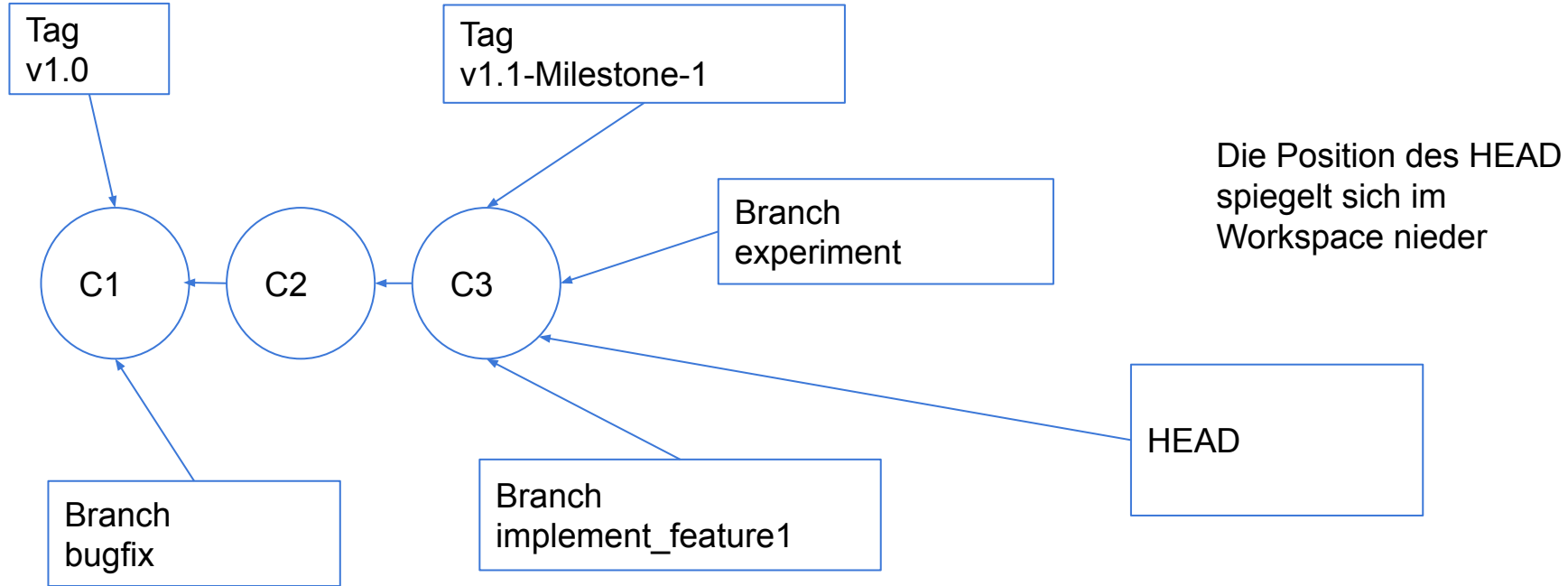


- Tags und Branches sind **trivial** in Erzeugung und Verwaltung
  - `git tag | branch new_name`
    - Erzeugung
  - `git tag | branch -d name`
    - Löschen
  - `git tag | branch --list`
    - Liste
  - `git branch -m old_name new_name`

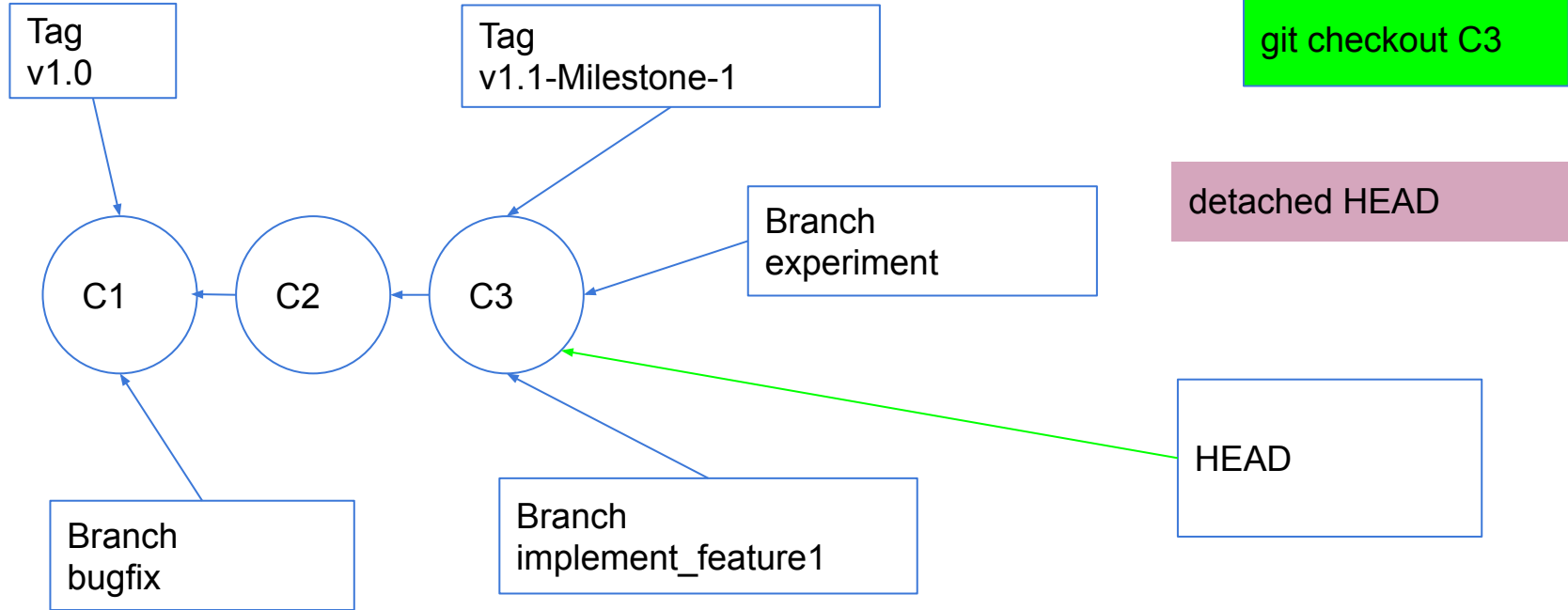
# Tags und Commits: Beispiel



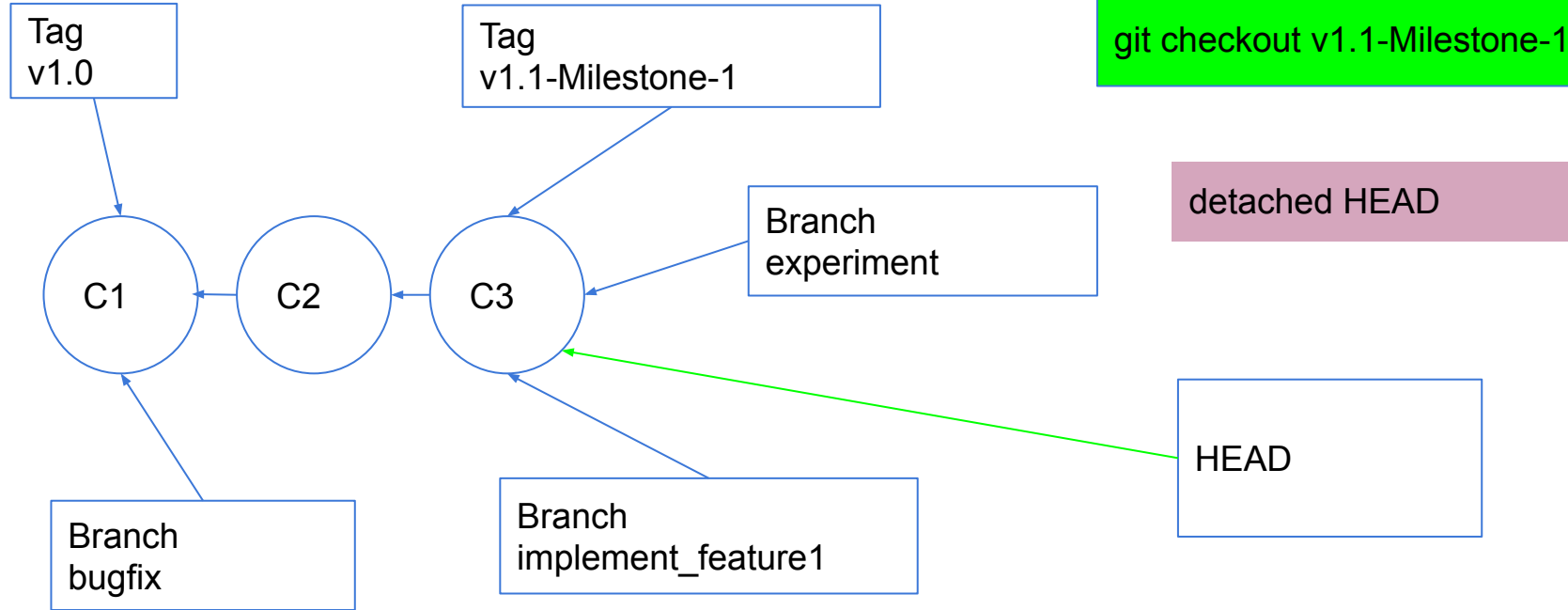
Tags und Branches  
sollen einen Überblick  
über den Stand des  
Projektes liefern



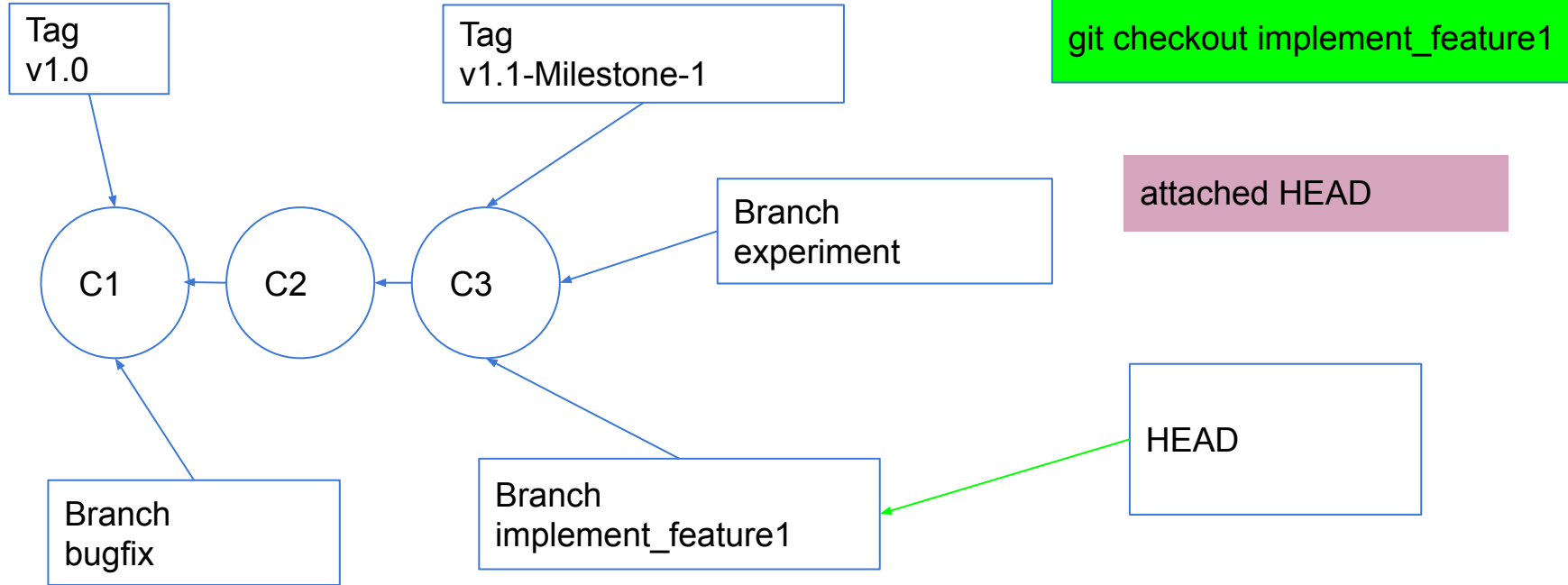
# Bewegen des HEAD: git checkout <hash>



# Bewegen des HEAD: git checkout <tag>



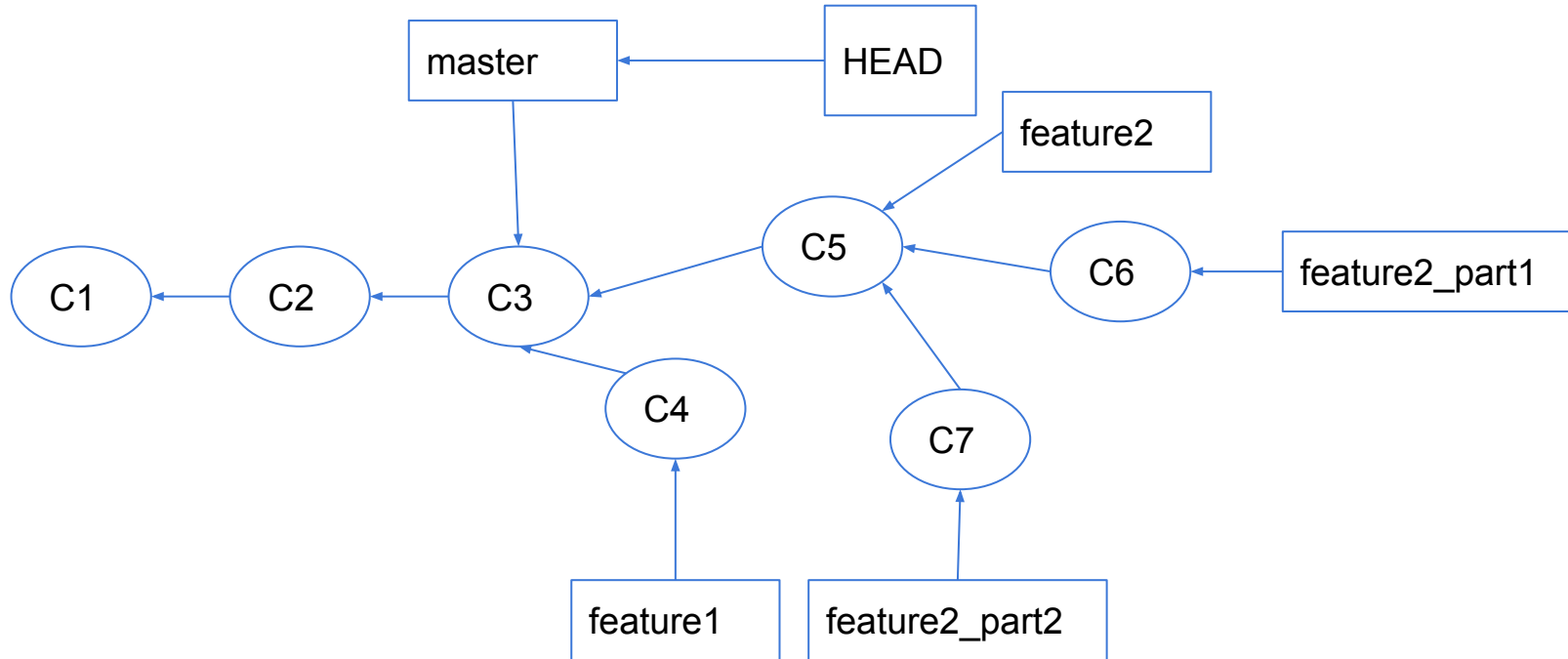
# Bewegen des HEAD: git checkout <branch>



- Ein checkout ändert den Zustand des Workspaces
- Best Practice
  - Checkout nur in unauffälligem Status
  - Später: Wie arbeite ich bei einem auffälligem Status
    - `checkout -f`
    - `reset`
    - Stashing
    - Work in Progress-Branch

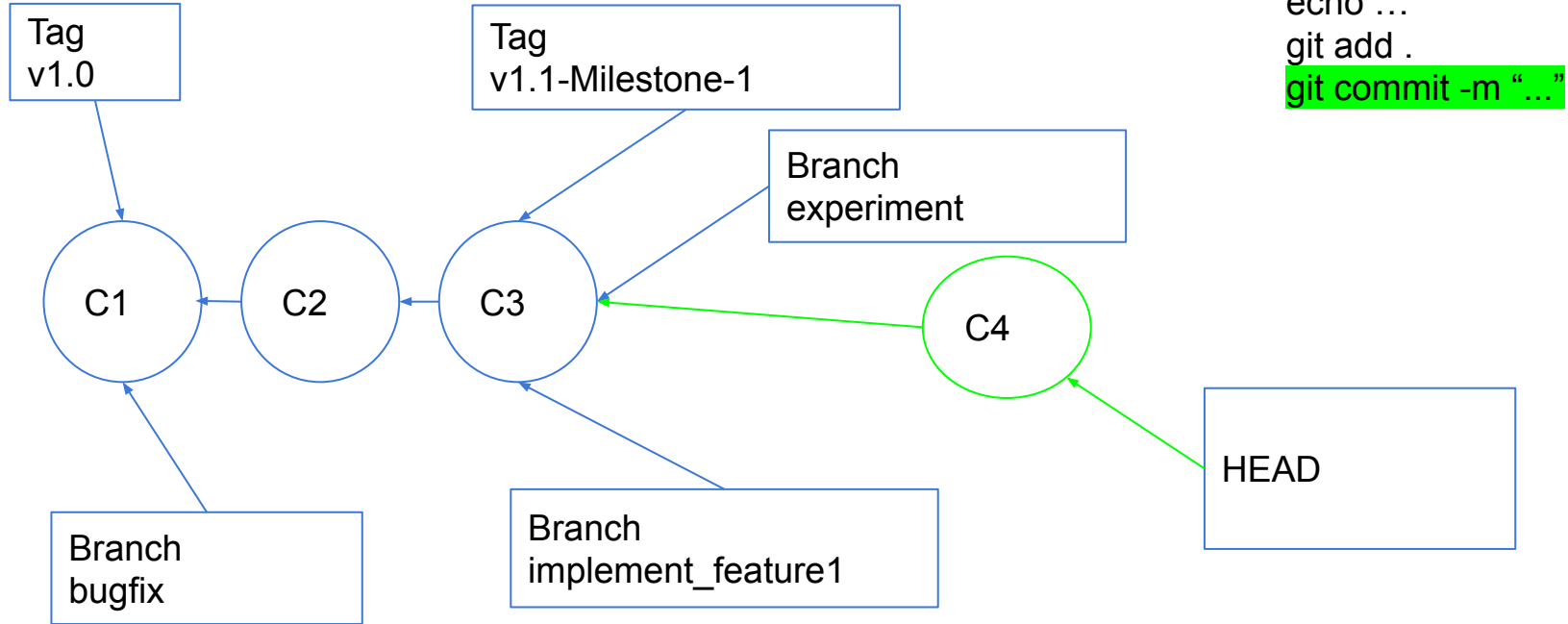
- `git log --oneline --decorate --all --graph`
- `git config --global alias.pl "log --oneline --decorate --all --graph"`



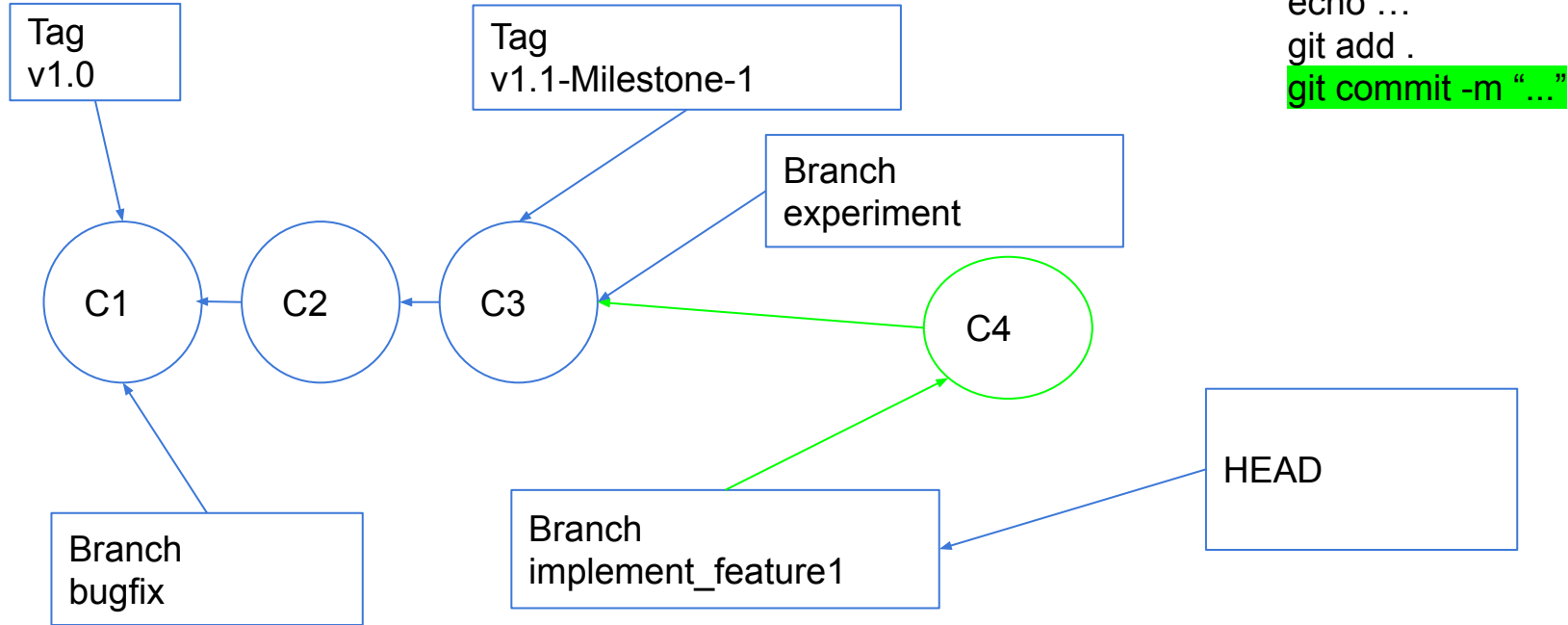


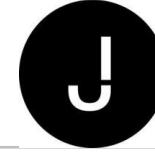
- Kopieren Sie das Skript, das Ihnen diese Struktur bereitstellt
- Visualisieren und Verifizieren Sie dieses Struktur durch den “pretty log”-Alias-Befehl
- checkout mit sauberem Status
  - Attached HEAD versus detached HEAD
    - Ausgabe des checkout-Befehls
    - git status
- Weitere Branches und Tags anlegen
  - git branch new\_name
    - Vorsicht: Der HEAD ist nicht an diesen neu erzeugter Branch attached!
  - git checkout -b new\_branch <hash> | <tag> | <branch>
- Branches, die nicht alleine an der Spitze einer Reihe stehen, können gelöscht werden: git branch -d <name>

# git commit: Detached HEAD

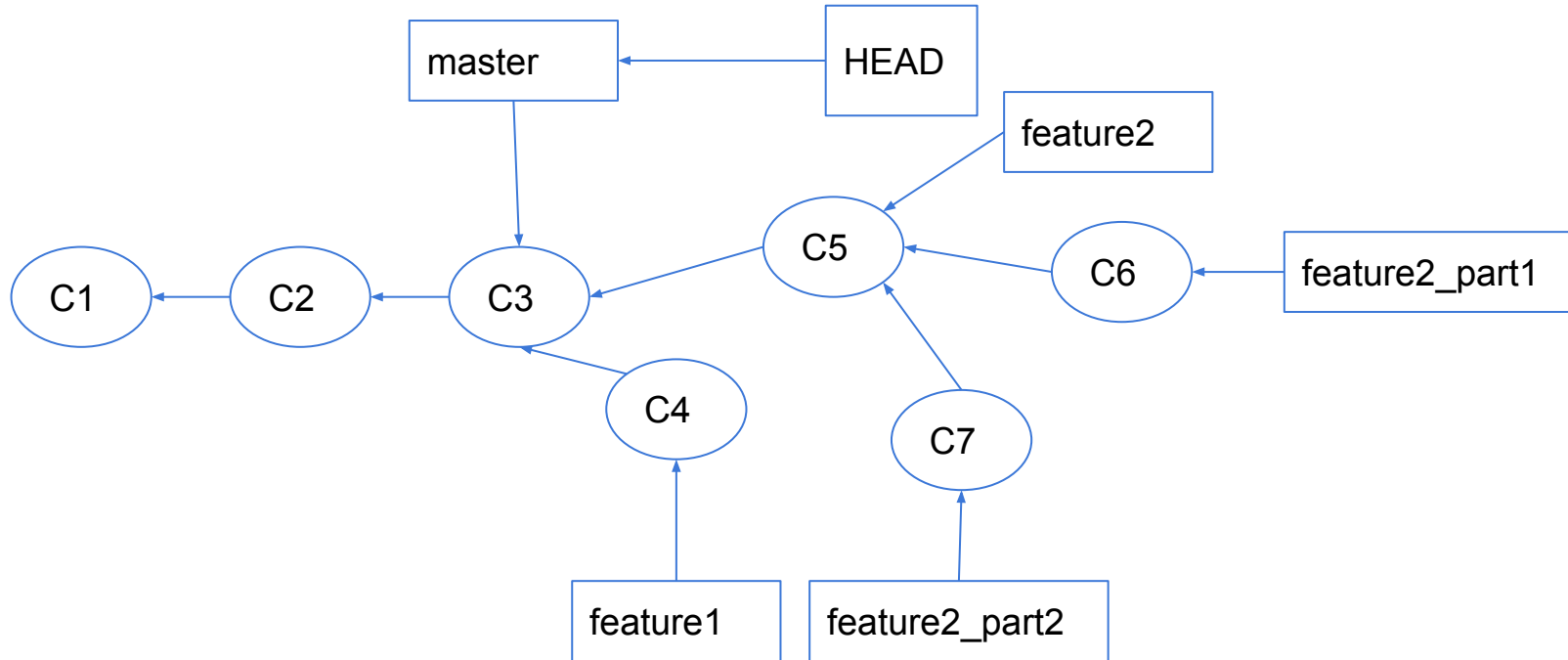


# git commit: Attached HEAD





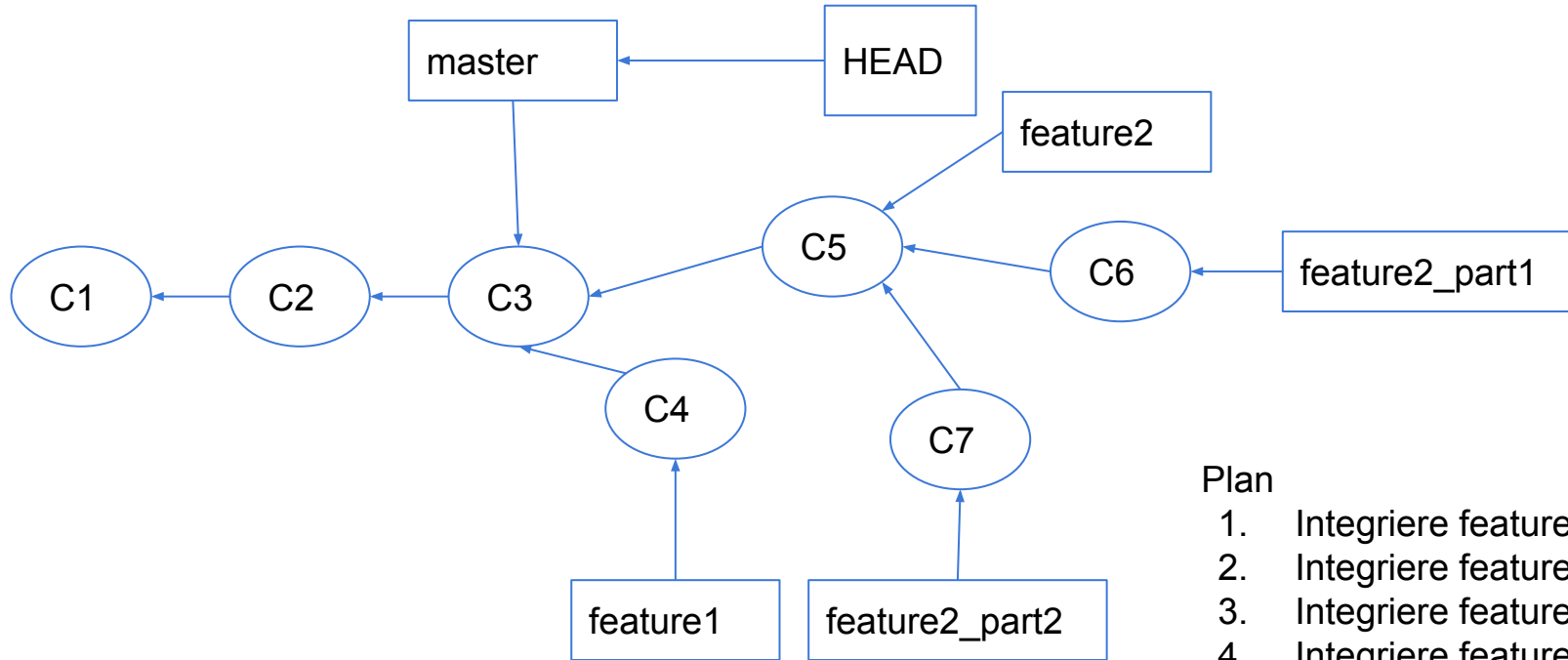
## Konsolidieren von Branches



- “Alle Änderungen, die in den feature-Banches eingeführt wurden, sollen in den master überführt werden”
  - Fachlich: Wir haben die features fertig entwickelt und wollen nun unseren Software-Stand weiterentwickeln
- Historie soll exakt die durchgeführten Änderungen widerspiegeln

Strategie: Merging

# Merge Plan

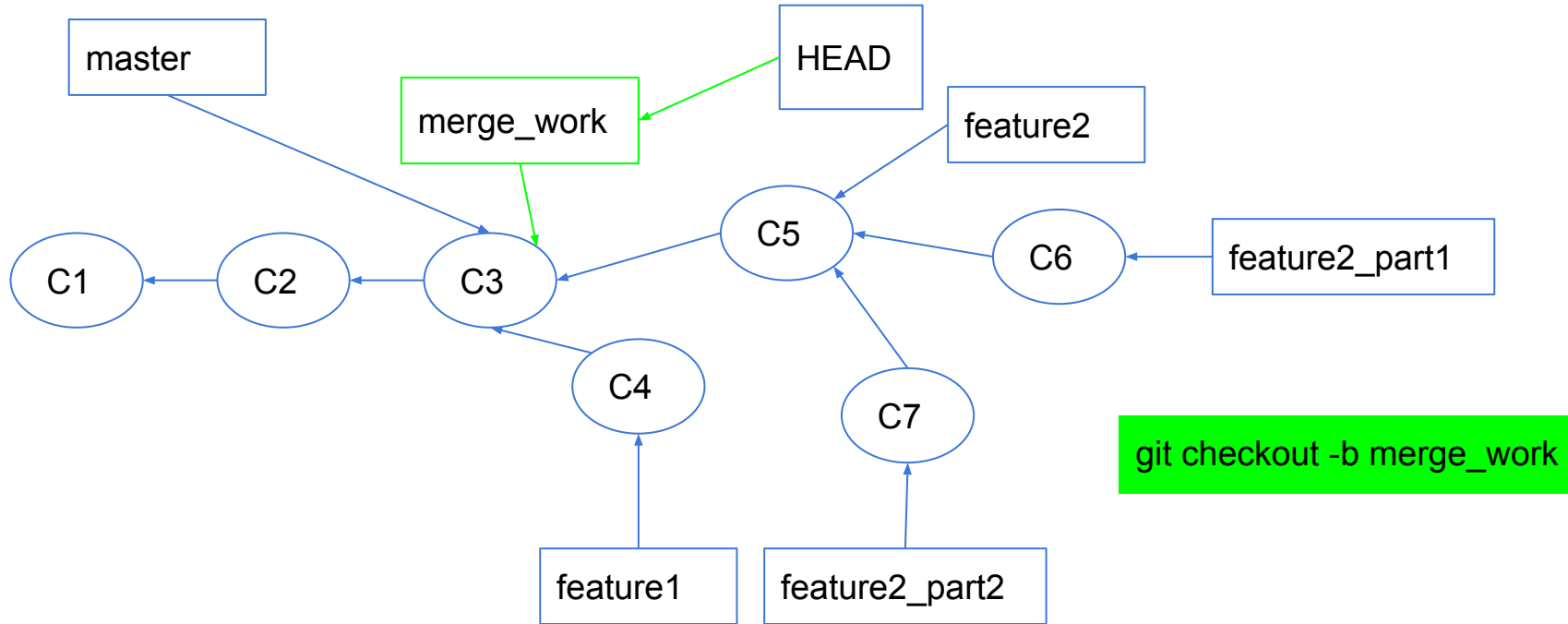


## Plan

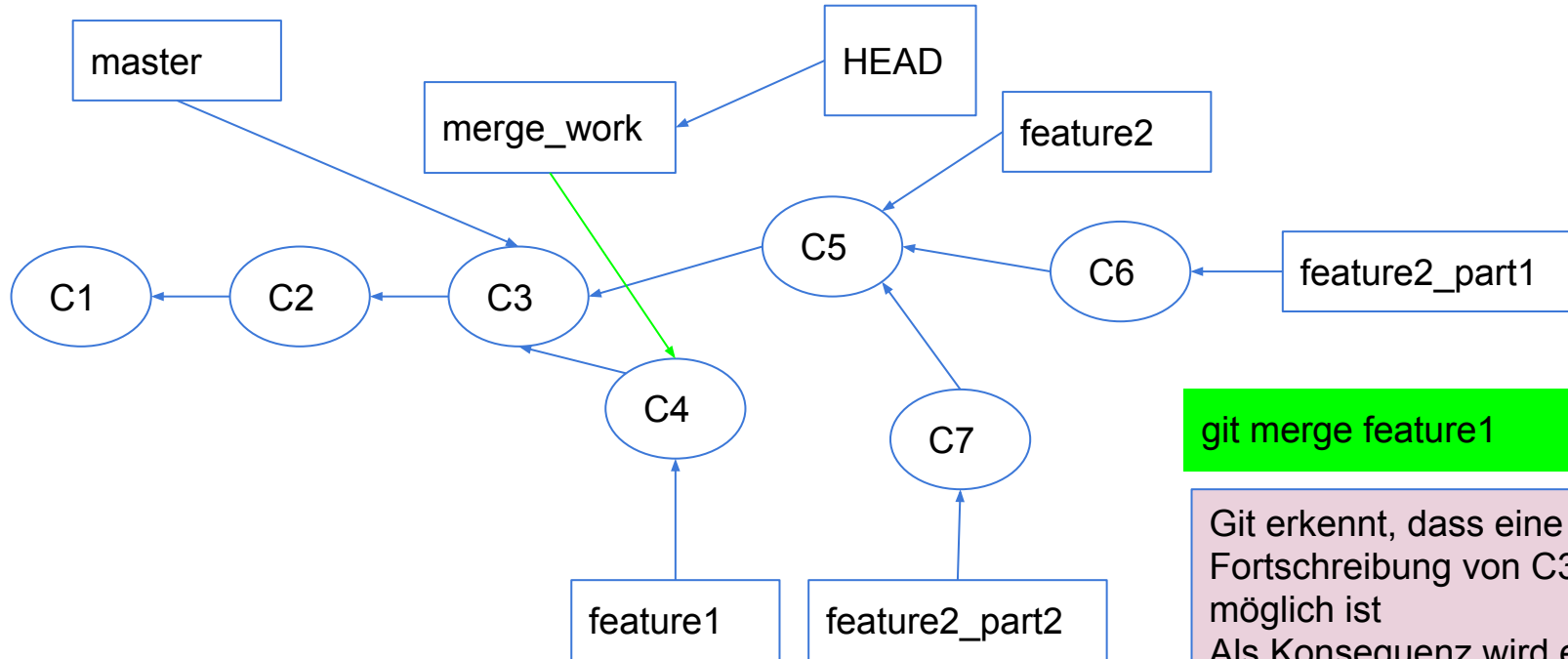
1. Integriere feature1
2. Integriere feature2
3. Integriere feature2\_part1
4. Integriere feature2\_part2



# Arbeiten Sie vorsichtig!



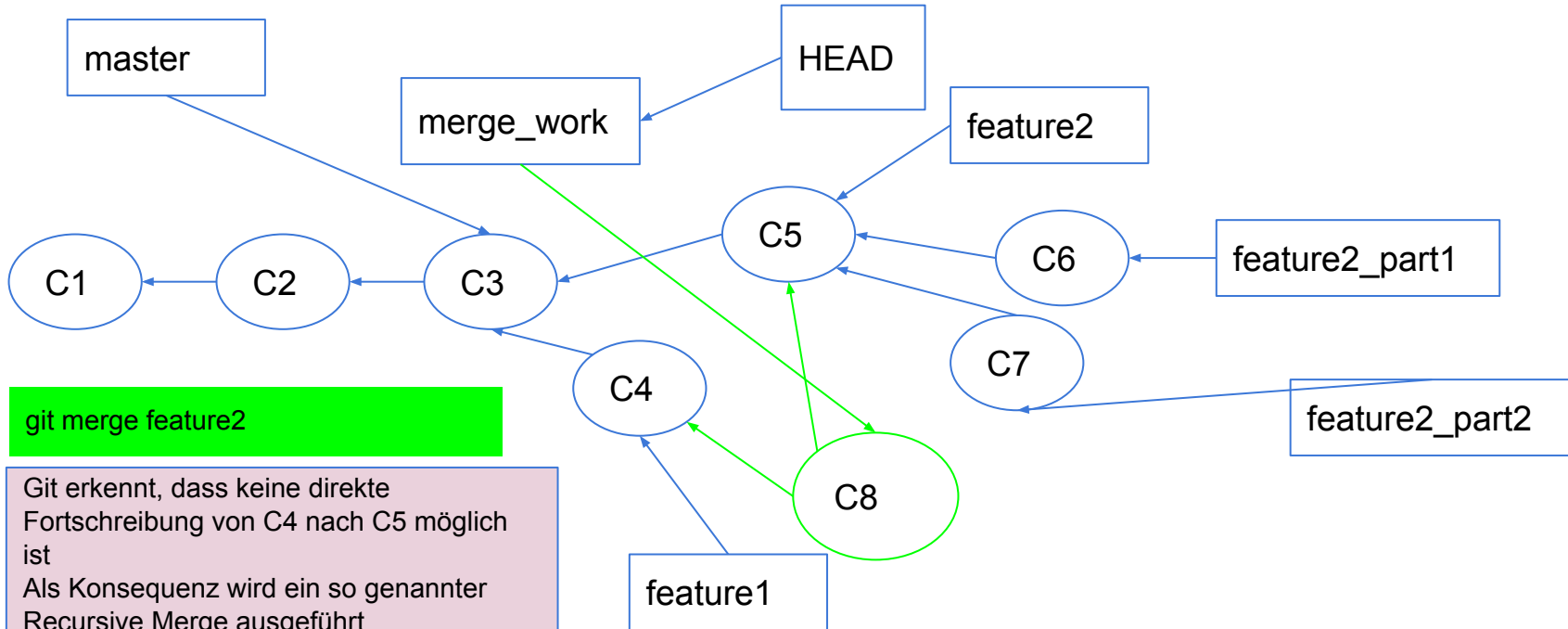
# Schritt 1: Integriere feature1 in merge\_work



**git merge feature1**

Git erkennt, dass eine direkte Fortschreibung von C3 nach C4 möglich ist  
Als Konsequenz wird ein so genannter Fast Forward ausgeführt  
Merge-Konflikte durch inkonsistente Änderungen sind unmöglich!

# Schritt 2: Integriere feature2 in merge\_work

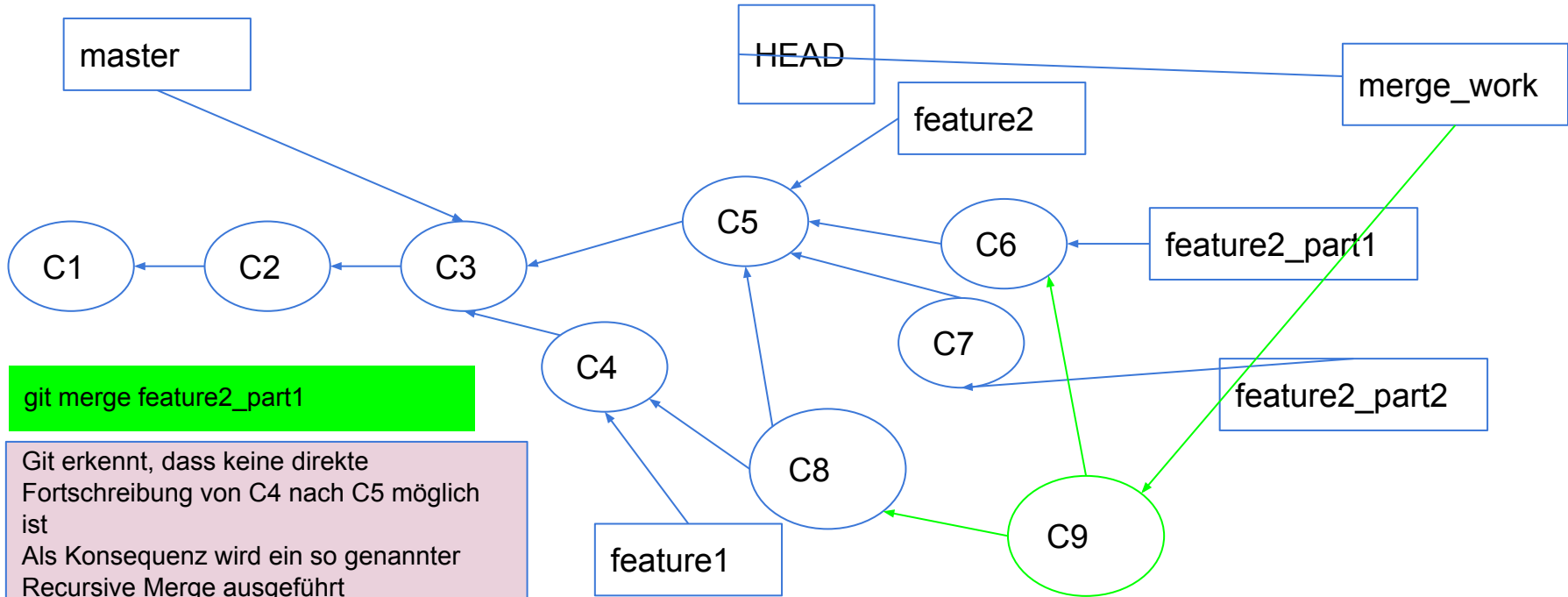


git merge feature2

Git erkennt, dass keine direkte Fortschreibung von C4 nach C5 möglich ist  
Als Konsequenz wird ein so genannter Recursive Merge ausgeführt  
Merge-Konflikte durch inkonsistente Änderungen sind immer möglich!  
Git erkennt solche Konflikte sicher + beinhaltet eine Auto Resolution

- Im Standard-Merging werden
  - Änderungen an unterschiedlichen Dateien
  - In unterschiedlichen Zeilen derselben Datei
- nicht als Konflikt signalisiert, sondern automatisch “behoben”
- VORSICHT
  - Das muss unbedingt im Rahmen des Mergens verifiziert/geprüft/getestet werden

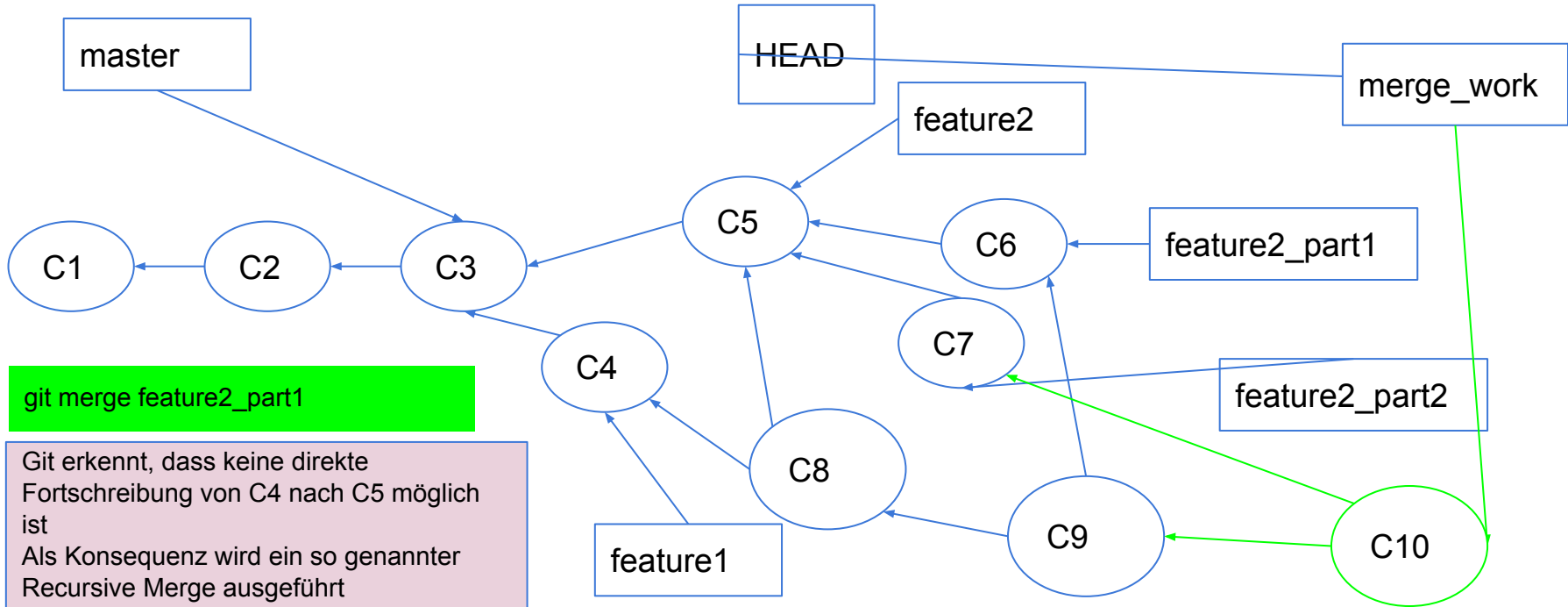
# Schritt 3: Integriere feature2\_part1 in merge\_work



**git merge feature2\_part1**

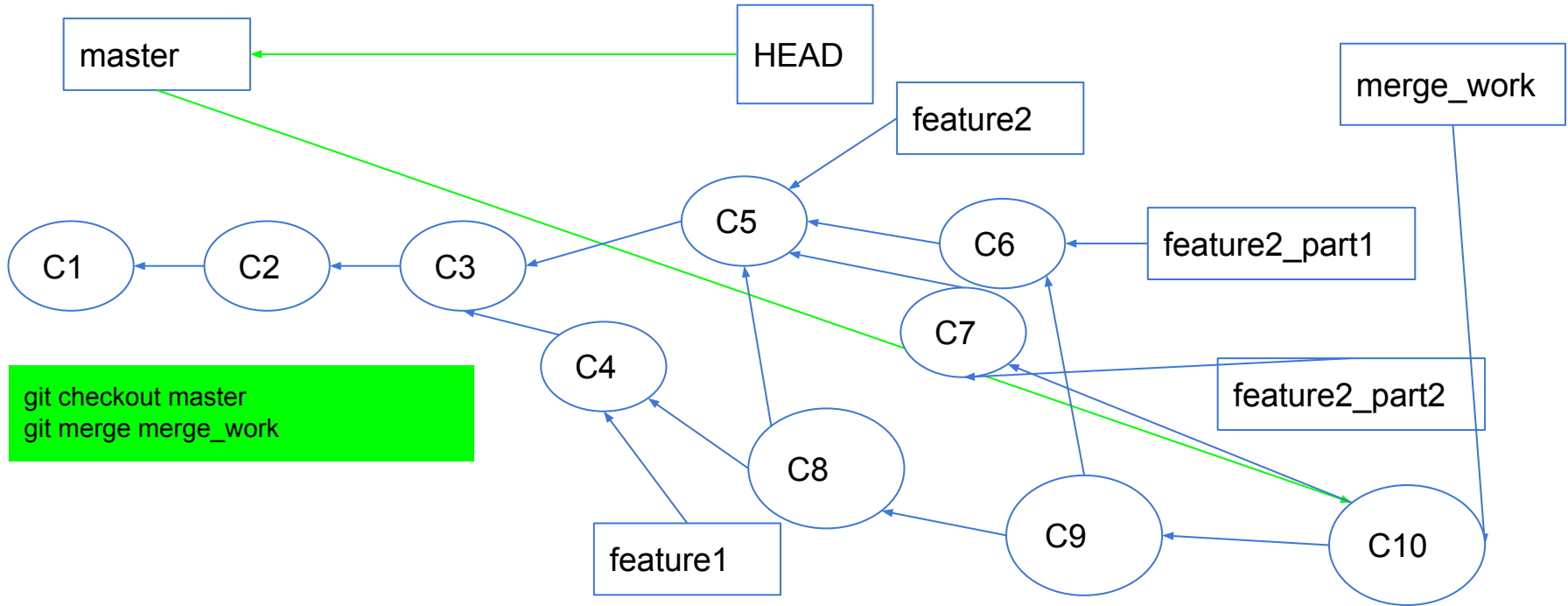
Git erkennt, dass keine direkte Fortschreibung von C4 nach C5 möglich ist  
Als Konsequenz wird ein so genannter Recursive Merge ausgeführt  
Merge-Konflikte durch inkonsistente Änderungen sind immer möglich!  
Git erkennt solche Konflikte sicher + beinhaltet eine Auto Resolution

# Schritt 4: Integriere feature2\_part2 in merge\_work

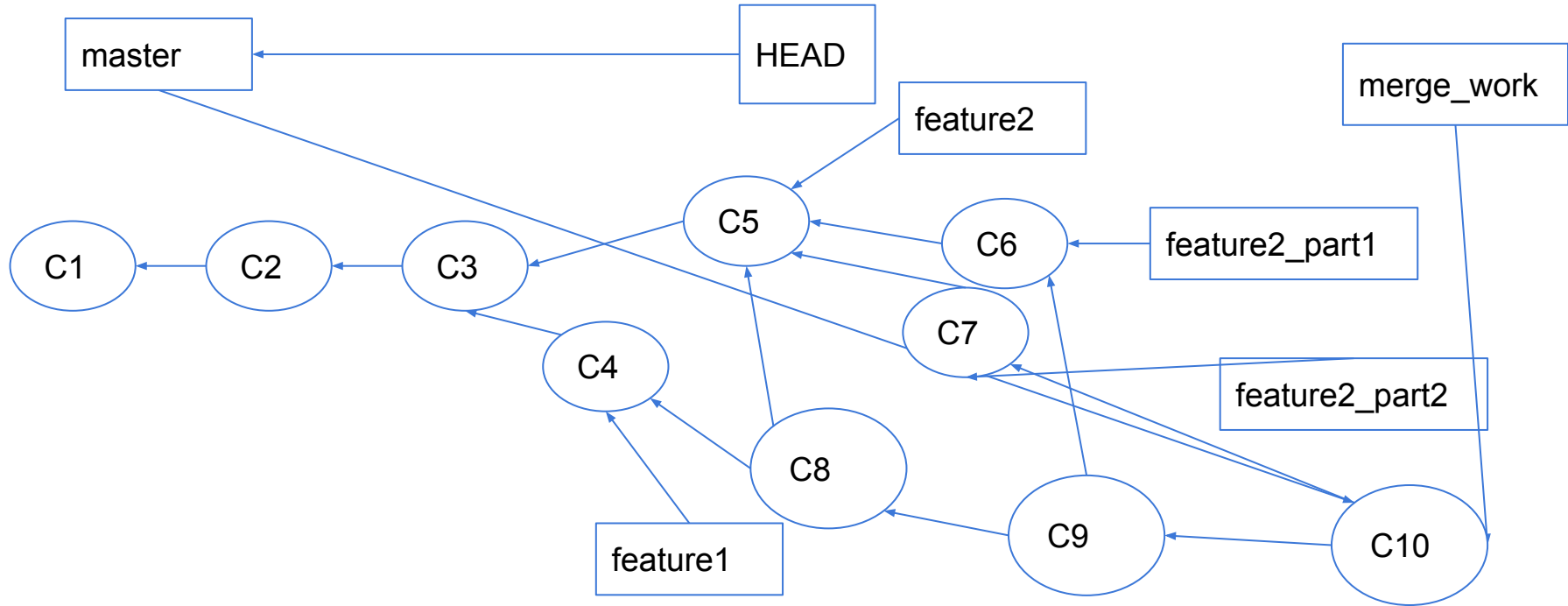


Git erkennt, dass keine direkte Fortschreibung von C4 nach C5 möglich ist  
Als Konsequenz wird ein so genannter Recursive Merge ausgeführt  
Merge-Konflikte durch inkonsistente Änderungen sind immer möglich!  
Git erkennt solche Konflikte sicher + beinhaltet eine Auto Resolution

# Schritt 5: Vorziehen des master

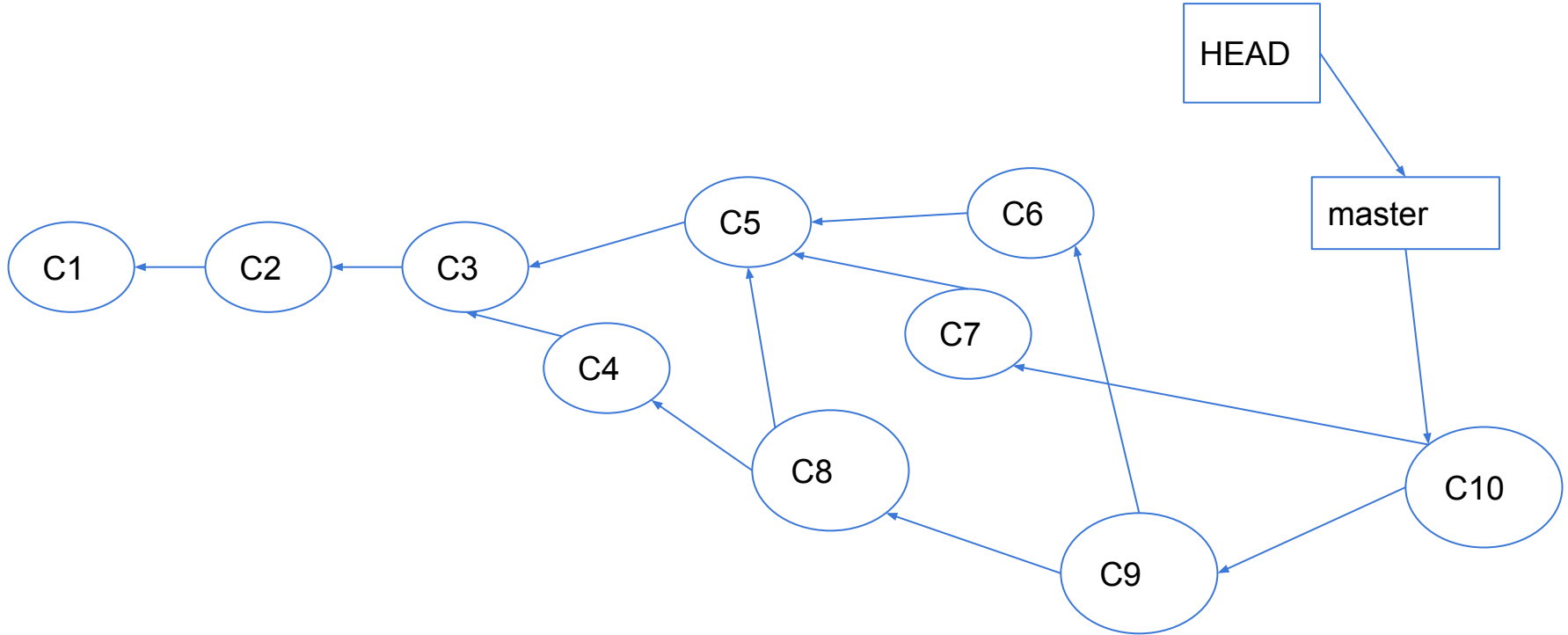


# Schritt 6: Aufräumen (optional)

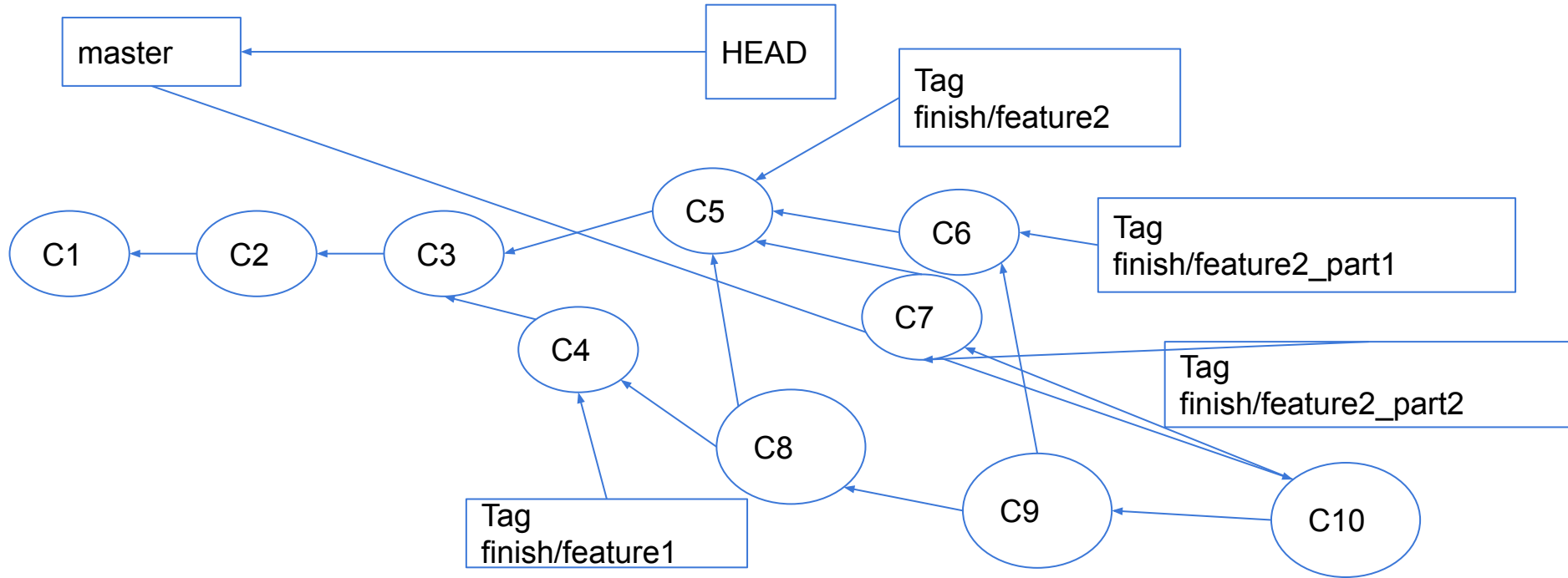


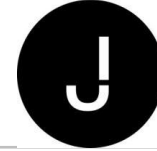


# Schritt 6: Aufräumen (gnadenlos)



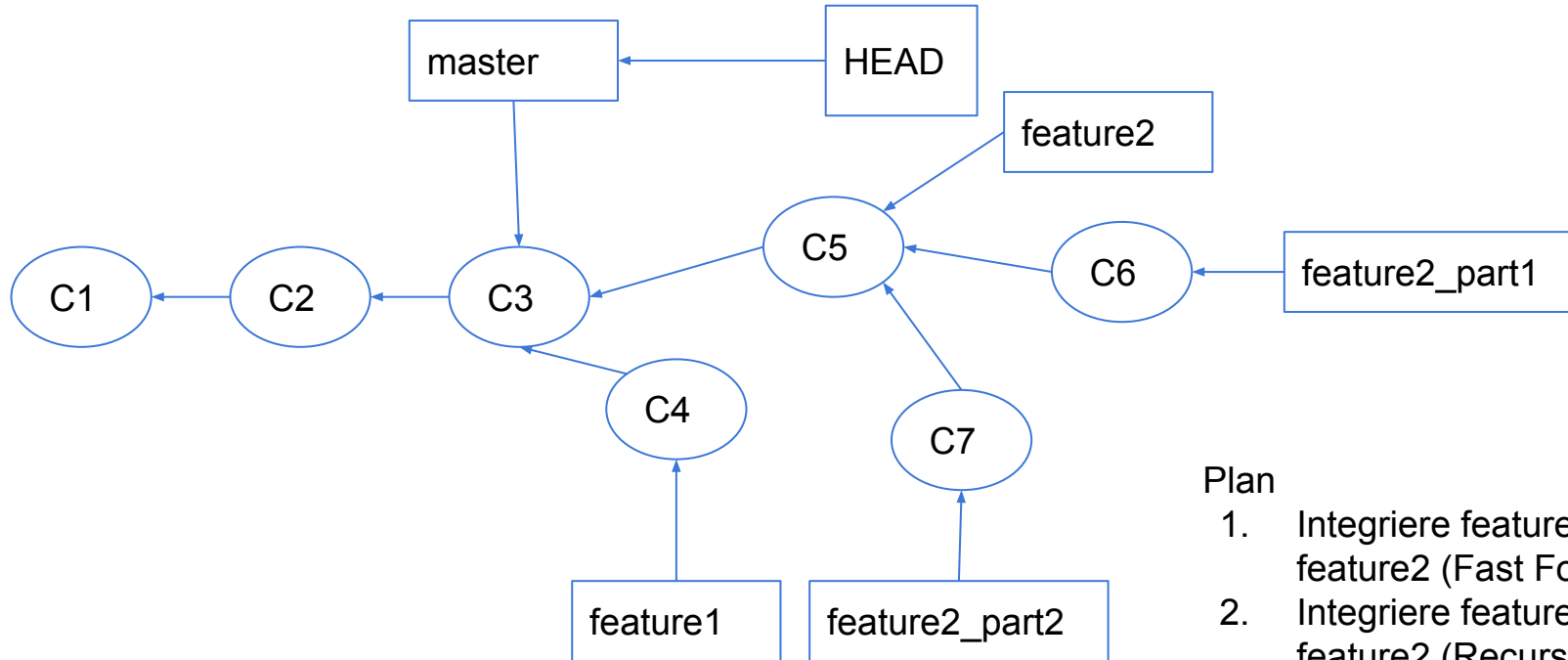
# Schritt 6: Aufräumen





```
rainer@rainer-Aspire-VN7-572G:~/git_training/training_branches$ git pl
* d9d19eb (HEAD -> master, merge_work) merge feature2_part2 and resolve conflicts
* eade7be change content-feature2, part2
* | 9cf5480 Merge branch 'feature2_part1' into merge_work
* | 5934ead change content-feature2, part1
* | 1b439be Merge branch 'feature2' into merge_work
* | b3e53d0 add content-feature2
* | 6f6fa51 add content-feature1
* 7a5f3ef change content
* 41074d1 add content
* 752bdf0 setup project
```

# ToDo: Merge Plan



## Plan

1. Integriere feature2\_part1 in feature2 (Fast Forward)
2. Integriere feature2\_part2 in feature2 (Recursive MIT conflict)
3. Integriere feature1 in feature2 (Recursive mit Auto Resolution)