JAVACREAM

Training
Consulting
Projectmanagement

GIT

Vorstellungsrunde

JAVACREAM
Training
Consulting
Projectmanagement

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

Einführung

Erwartungshaltung an Versionsverwaltungssystem

- 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

Realisierung mit GIT

- 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

Realisierung mit GIT, Git-Werkzeuge

- 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

Zu den Git-Werkzeugen

- Desktop
  - TortoiseGit
  - SourceTree
  - …
- PlugIns für Editoren
  - Eclipse
  - Visual Studio
  - Visual Studio Code
  - ...

Realisierung mit GIT, Git-Werkzeuge, Git Server

- 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

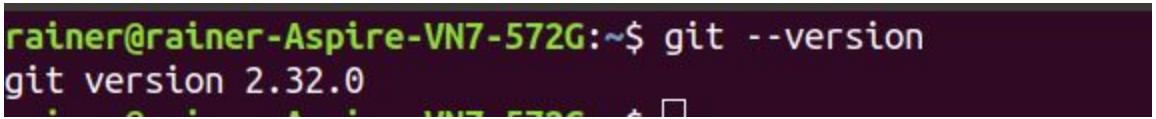
Git Server

- 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

Git Bash

- 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
  -



Der erste Git-Befehl: config

- 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-Projektverzeichnisses


```
mkdir first
cd first
```

```
git init
```

```
git status
        Fehlerfrei
```

Normales Verzeichnis -> Git-Projektverzeichnis

Git Workspace

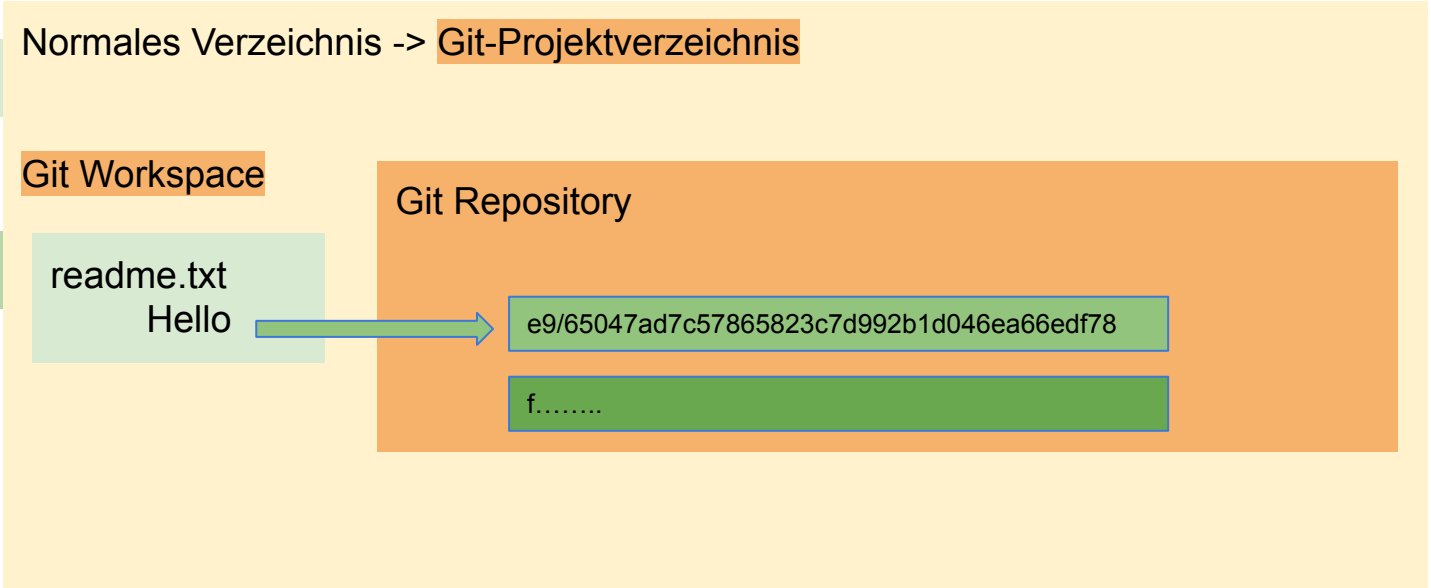
Git Repository

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

Hinzufügen von Informationen


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

```
git add readme.txt
```



Exkurs

- 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”

Interpretation

- 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

Commits definieren Stände


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

```
git add readme.txt
```

```
git commit -m “warum?”
```

Normales Verzeichnis -> Git-Projektverzeichnis

Git Workspace

readme.txt
Hello

Git Repository

Index / Staging Area

e9/65047ad7c57865823c7d992b1d046ea66edf78

f……..

Commit

Objekt

Aufbau des Commit-Objekts

- 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

Interpretation

- 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)

Objekte in Git

- 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

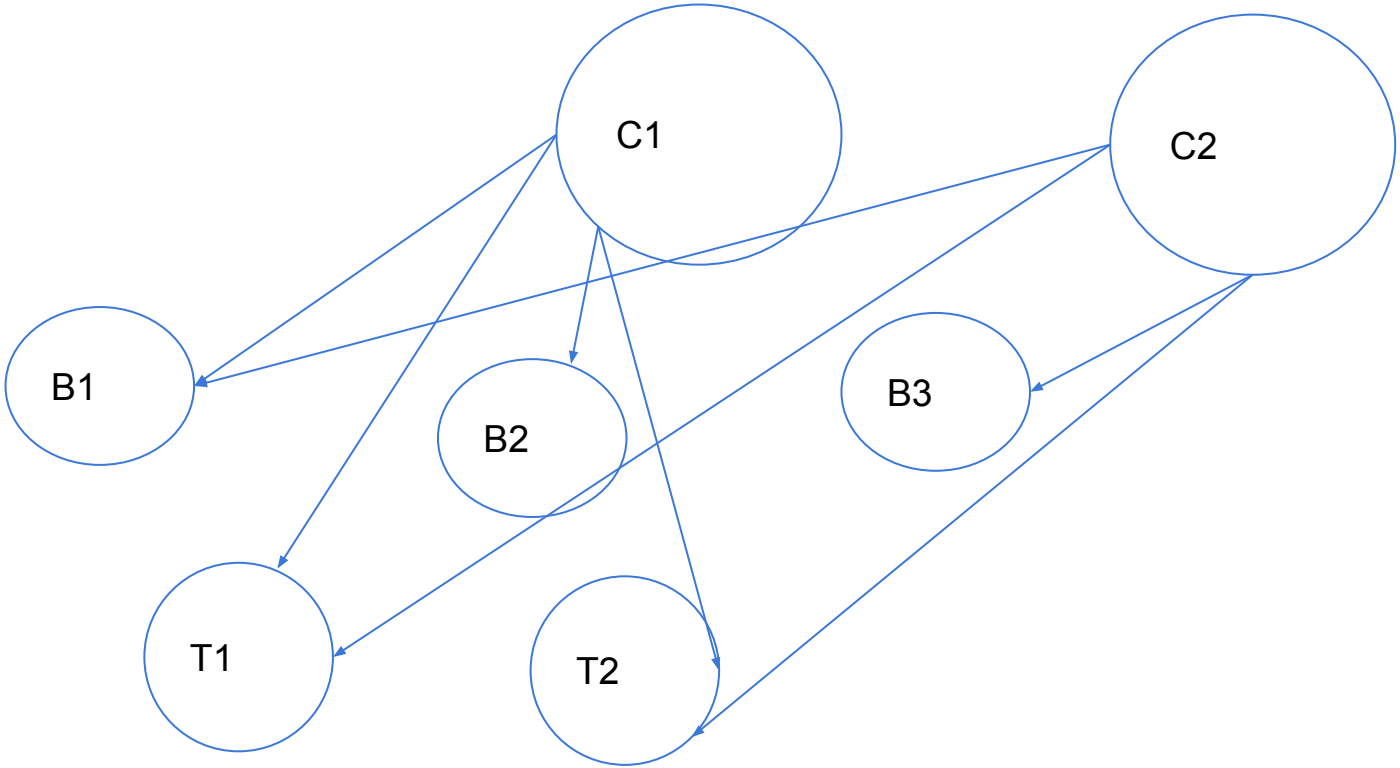
Arbeiten mit git

- 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

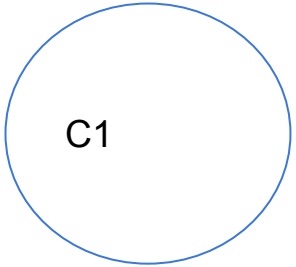
Exkurs: .gitignore

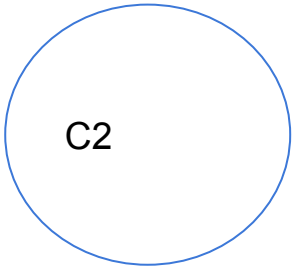
- Textdatei als Bestandteil des Workspaces
  - oder eines Unterverzeichnisses
- In dieser Textdatei werden Regeln hinterlegt, die Dateien ausschließen
  - Unterverzeichnis-.gitignores werden gemerged mit denen der Ober-Verzeichnisse
- Hinweis
  - Standard-Namen (*.bak, …) werden automatisch ausgeschlossen

Die Git-Objekte im Detail



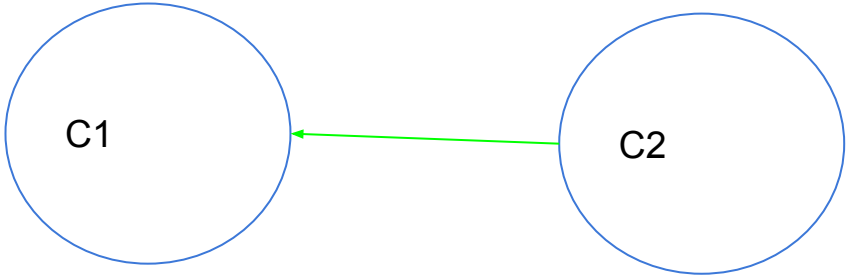
Weitere Visualisierung des Repos





Es werden nur noch die Commit-Objekte gezeichnet

Weitere Visualisierung des Repos:
Korrekt



Aufbau des Commit-Objekts

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

Interpretation

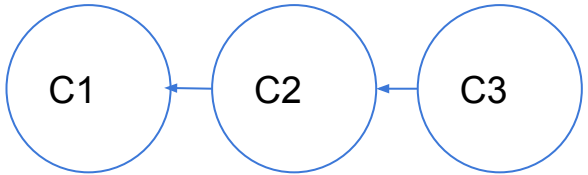
- Ist in einem Repository ein Commit mit dem Hash 9e7f4ac1208a5ceaa5f36e4d4a96276b9318c5fa vorhanden:
  - Rainer Sawitzki hat am um eine Commit erstellt mit der Message ... und den Dateien … ausgehend vom Commit mit dem Hashwert …. der dann wiederum den Hash ee23e95a40724fb1ad5b119d2ed9e8f7c069813e mit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

Git-Technologie = Blockchain

- 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

Aktueller Stand

Fokus auf Hash-Werte, “Nerd-Modus”



Pragmatisches Arbeiten: Alias-Namen auf Commit-Objekte

- 2 Szenarien für die Einführung eines Namens
  - Definition eines fixen Standes
    - Typischerweise Versionen
      - v1.0, v1.1-Milestone1
    - “HeuteMorgen”
  - Bennenung 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”, ...

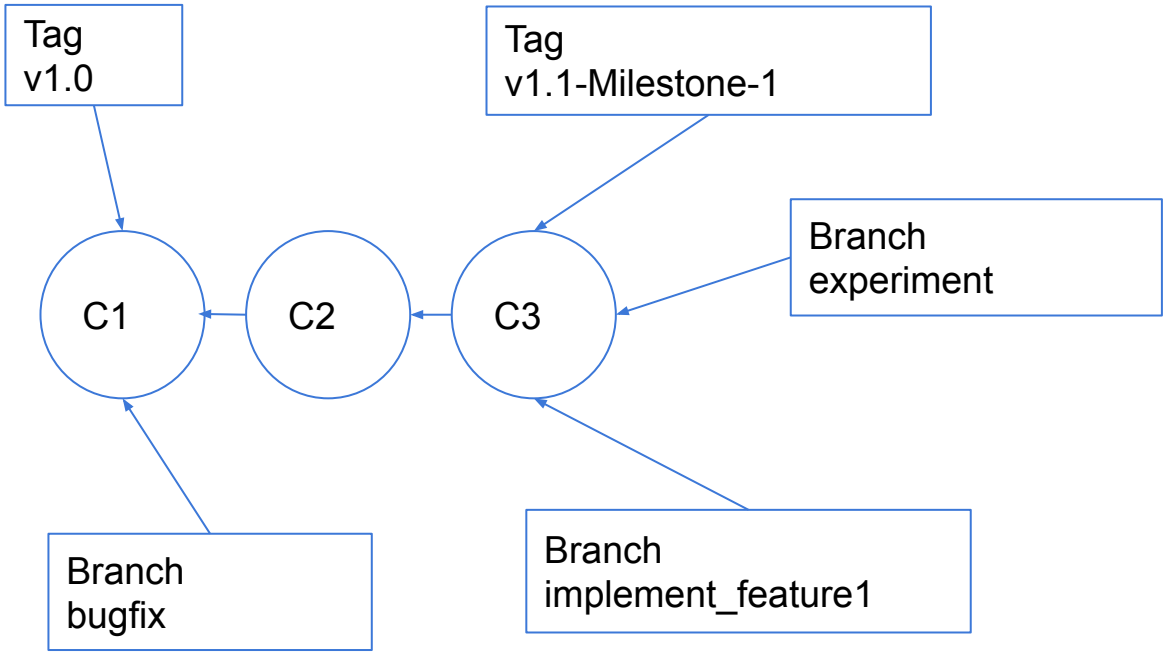
Pragmatisches Arbeiten: Alias-Namen auf Commit-Objekte mit Git

- 2 Szenarien für die Einführung eines Namens
  - Definition eines fixen Standes
    - Typischerweise Versionen
      - v1.0, v1.1-Milestone1
    - “HeuteMorgen”
    - Git: Tag
  - Bennenung 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

Umsetzung in Git

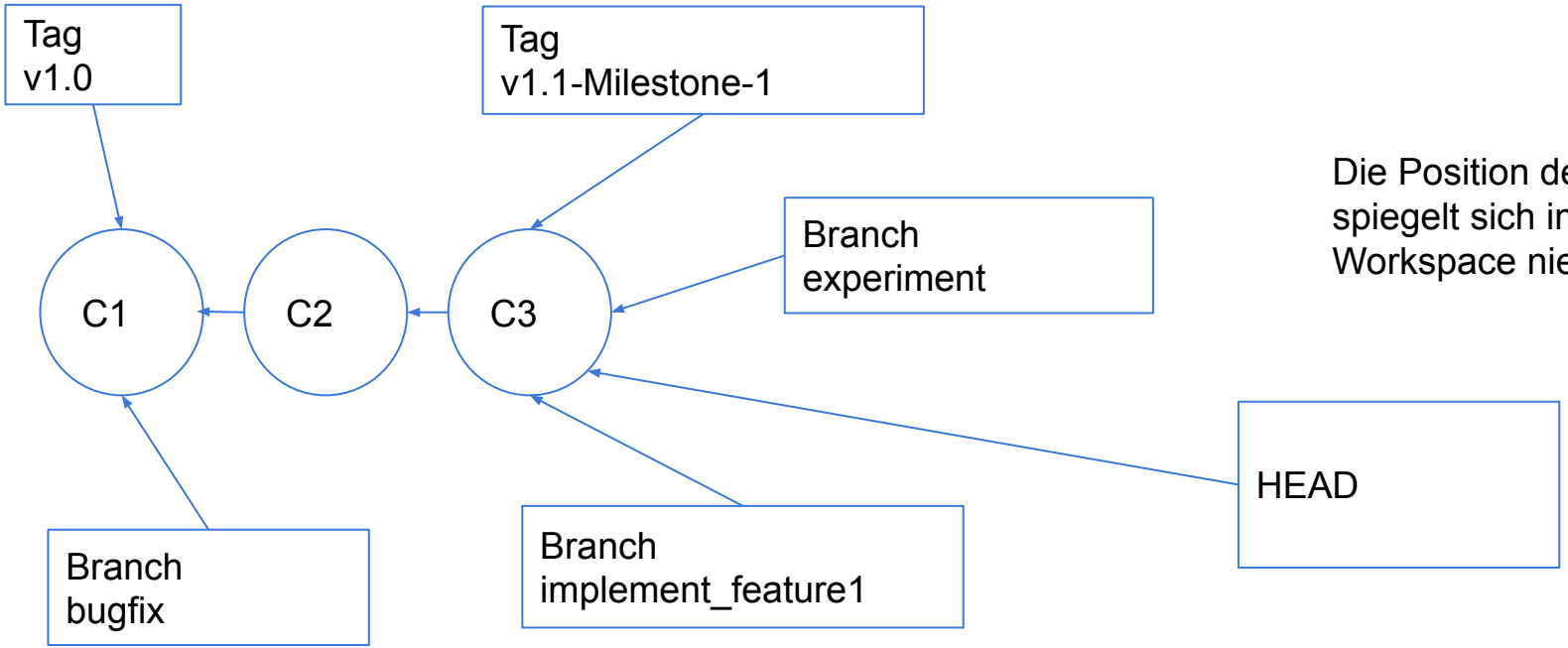
- Tags und Branches sind **trivial** in Erzeugung und Verwaltung
  - git tag | branch new_name
    - Erzeugung
  - git tag | branch -d name
    - Löschen
  - git tag | branch --list
    - Liste
  - git branch -m old_name new_name

Tags und Commits: Beispiel



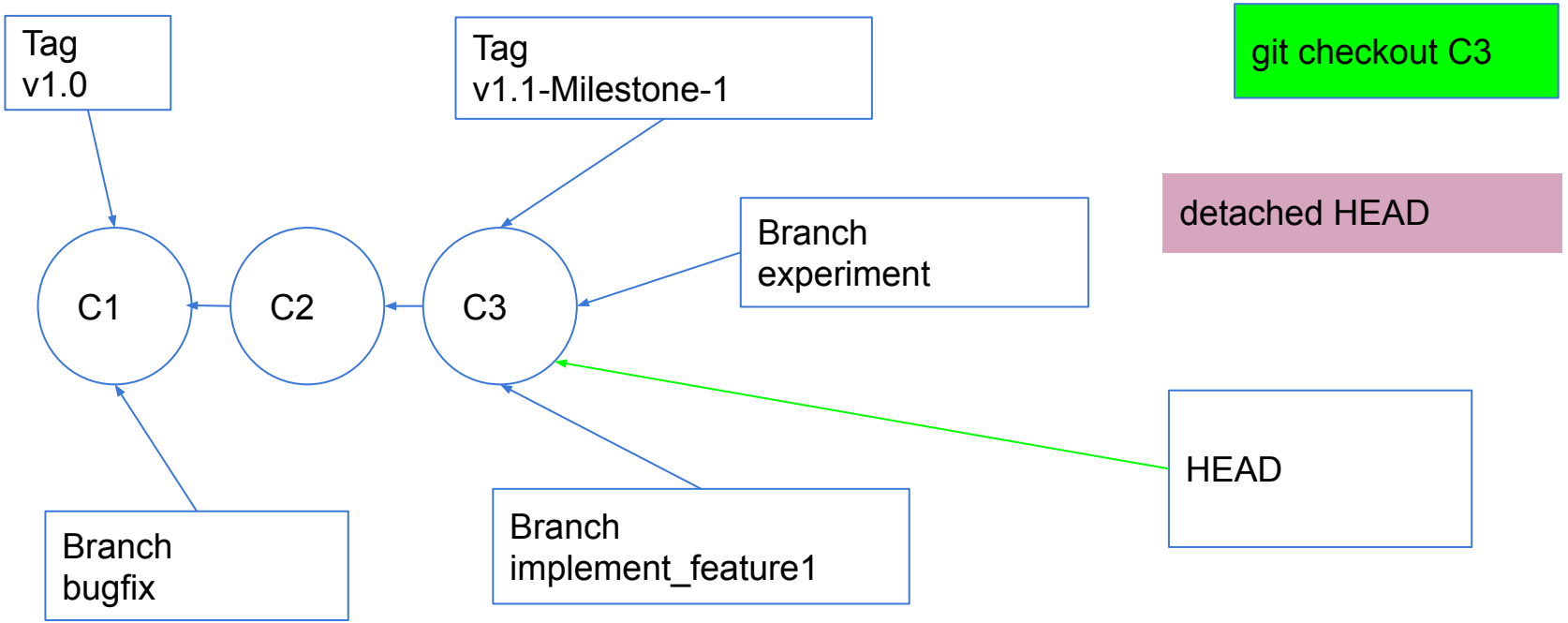
Tags und Branches sollen einen Überblick über den Stand des Projektes liefern

Aktuelle Position: HEAD

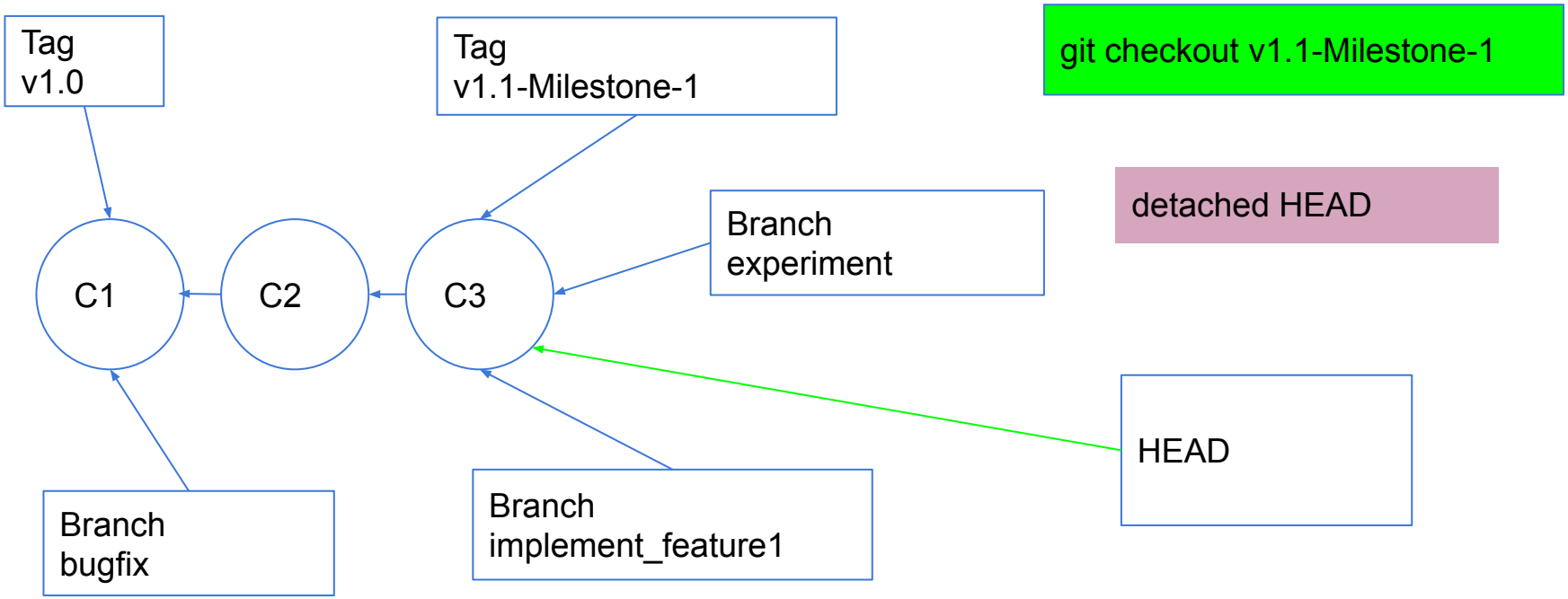


Die Position des HEAD spiegelt sich im Workspace nieder

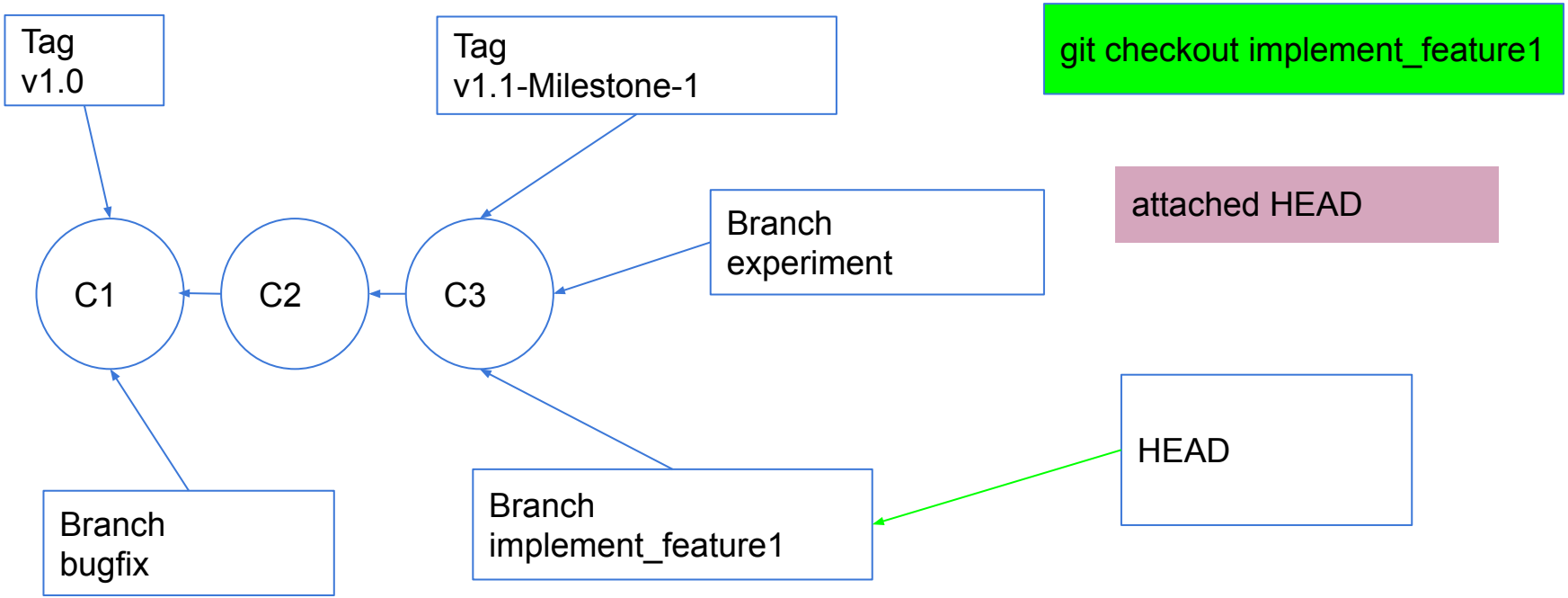
Bewegen des HEAD: git checkout <hash>



Bewegen des HEAD: git checkout <tag>



Bewegen des HEAD: git checkout <branch>



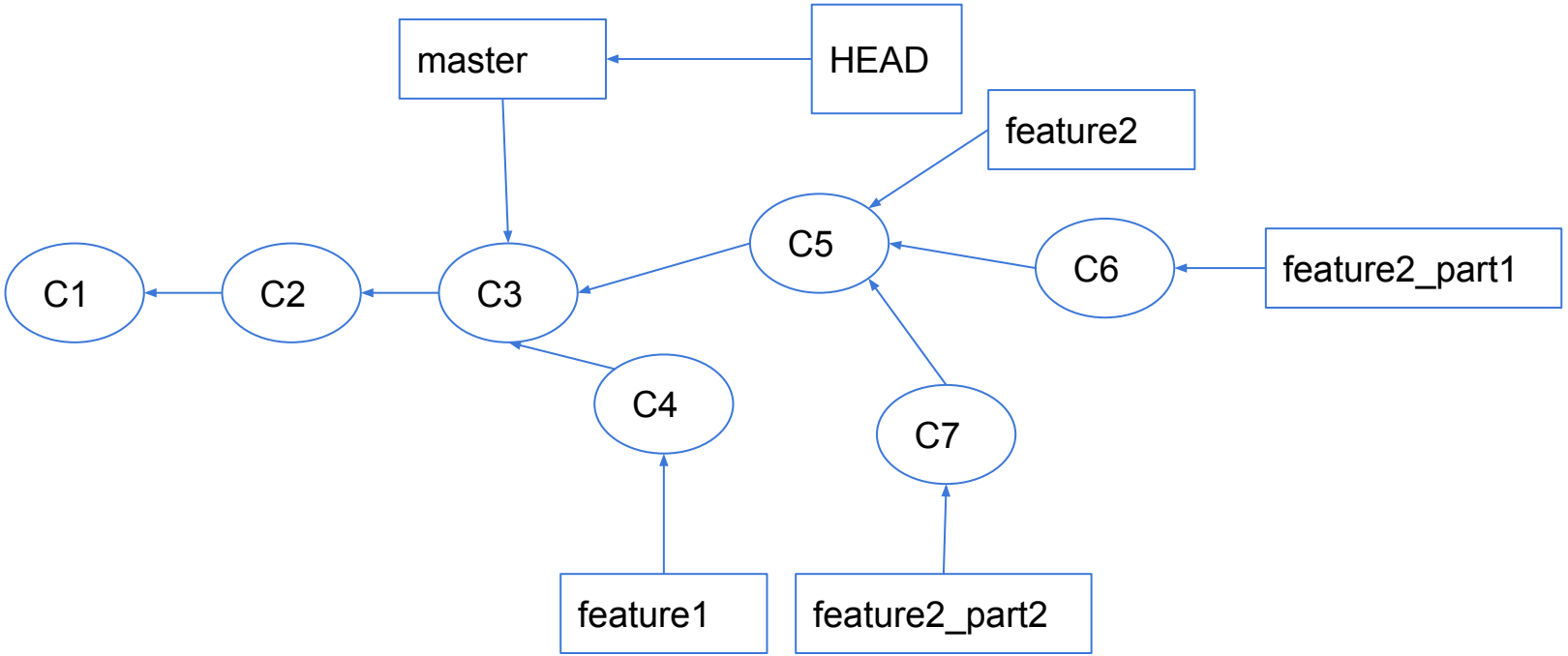
Bemerkung zum checkout

- 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

Exkurs: Command-Aliase

- git log --oneline --decorate --all --graph
- git config --global alias.pl “log --oneline --decorate --all --graph”

training branches



ToDo

- 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>