JAVACREAM

Training
Consulting
Projectmanagement

GIT

Kurze Vorstellung

- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemstellung
- Konkrete individuelle Zielsetzung

Einführung

Allgemein: Versionsverwaltung

- Quellcode = “Werke” werden vom Versionsverwaltungssystem mit Meta-Informationen ergänzt
  - Wer hat wann warum welche Änderungen gemacht?
- Der Stand eines Projekts wird durch ein Aggregat von Werken definiert
- Zentrale Ablage aller relevanten Informationen
  - inklusive Authentifizierung und Autorisierung
- Werkzeugen und Verfahren für die Team-Zusammenzusammenarbeit
- Werkzeuge für die Visualisierung und Konsolidierung der Arbeit

Git: Versionsverwaltung

- Quellcode = “Werke” werden vom Versionsverwaltungssystem mit Meta-Informationen ergänzt
  - Wer hat wann warum welche Änderungen gemacht?
- Der Stand eines Projekts wird durch ein Aggregat von Werken definiert
- Zentrale Ablage aller relevanten Informationen
  - inklusive Authentifizierung und Autorisierung
- Werkzeugen und Verfahren für die Team-Zusammenzusammenarbeit
  - Konzepte und Technik
- Werkzeuge für die Visualisierung und Konsolidierung der Arbeit
  - Konsolenbasiert

Git und Git Server-Lösungen: Versionsverwaltung

- Quellcode = “Werke” werden vom Versionsverwaltungssystem mit Meta-Informationen ergänzt
  - Wer hat wann warum welche Änderungen gemacht?
- Der Stand eines Projekts wird durch ein Aggregat von Werken definiert
- Zentrale Ablage aller relevanten Informationen
  - inklusive Authentifizierung und Autorisierung
  - Produkt-Lösungen, z.B. **GitHub** (Microsoft), GitLab (GitLab.com), BitBucket (Atlassian)
- Werkzeugen und Verfahren für die Team-Zusammenzusammenarbeit
  - Konzepte und Technik
  - **Pull** / Merge **Requests**
- Werkzeuge für die Visualisierung und Konsolidierung der Arbeit
  - Konsolenbasiert
  - Web Frontend

Git, Git Server-Lösungen und Client-Werkzeuge: Versionsverwaltung

- Quellcode = “Werke” werden vom Versionsverwaltungssystem mit Meta-Informationen ergänzt
  - Wer hat wann warum welche Änderungen gemacht?
- Der Stand eines Projekts wird durch ein Aggregat von Werken definiert
- Zentrale Ablage aller relevanten Informationen
  - inklusive Authentifizierung und Autorisierung
  - Produkt-Lösungen, z.B. **GitHub** (Microsoft), GitLab (GitLab.com), BitBucket (Atlassian)
- Werkzeugen und Verfahren für die Team-Zusammenzusammenarbeit
  - Konzepte und Technik
  - **Pull** / Merge **Requests**
- Werkzeuge für die Visualisierung und Konsolidierung der Arbeit
  - Konsolenbasiert
  - Web Frontend
  - Entwicklungsumgebungen (IntelliJ, Visual Studio Code) native-Git-Installationen mit Turtoise, …

Git, Git Server-Lösungen und Client-Werkzeuge: Versionsverwaltung

- Quellcode = “Werke” werden vom Versionsverwaltungssystem mit Meta-Informationen ergänzt
  - Wer hat wann warum welche Änderungen gemacht?
- Der Stand eines Projekts wird durch ein Aggregat von Werken definiert
- Zentrale Ablage aller relevanten Informationen
  - inklusive Authentifizierung und Autorisierung
  - Produkt-Lösungen, z.B. **GitHub** (Microsoft), GitLab (GitLab.com), BitBucket (Atlassian)
- Werkzeugen und Verfahren für die Team-Zusammenzusammenarbeit
  - Konzepte und Technik
  - **Pull** / Merge **Requests**
- Werkzeuge für die Visualisierung und Konsolidierung der Arbeit
  - Konsolenbasiert
  - Web Frontend
  - Entwicklungsumgebungen (IntelliJ, Visual Studio Code) native-Git-Installationen mit Turtoise, …

Seminar Tag 1 +x

Seminar Tag 2

First Contact

Installation

- Installiert wird das Executable “git”
  - Kein Client, der mit einem Server kommuniziert, sondern das komplette Core-Versionsverwaltungssystem
  - Kein Hintergrund-Dienst, Service, Daemon
    - Das Git-Executable stellt während der Kommando-Ausführung die Funktionalität eines Versionsverwaltungssystems zur Verfügung

Vorbereitung des Arbeitens mit Git

- ~~Server-Administrator legt einen Account für Sie an~~
- ~~Konfiguration: Server-URL~~

Git Core braucht keine Server

- Erstellung einer Git-Konfigurationsdatei (eine einfache Text-Datei namens .gitconfig in Ihrem User-Home)
  - Minimal:
    - user.name
    - user.email
  - git config --global user.name “Rainer Sawitzki”
  - git config --global user.email training@rainer-sawitzki.de
  -

Ein erstes Git Repository

- Ein Repository repräsentiert je nach Ihrer Projekt-Organisation
  - Ein komplettes Software-Projekt
  - ein Modul eines Software-Projektes
  - Eine Gruppe von Software-Projekten
- Auf der Ebene einer Entwickler-Maschine ist ein Git-Repository Bestandteil eines normalen Directories
- Anlegen im Seminar erst einmal komplett untypisch durch Initialisierung eines leeren Repositories
  - In der Realität: Clone eines Server-Repositories

Schritte

- mkdir first
  - Normales Arbeitsverzeichnis auf einer lokalen Maschine
- cd first
- git init
  - Legt das Git-Repository im Unterverzeichnis .git an
  - Das “normale Arbeitsverzeichnis” ist nun ein Git-Projekt-Verzeichnis
    - Alles andere als .git: “Git Workspace”
- Check
  - git status

Exkurs: Verteilte Datenverwaltung

- Problemstellung
  - Wie kann Information in einer Art und Weise verteilt werden, dass jede Kopie garantiert Informationen konsistent hält?
  - Konkrete mit den Begriffen der Versionsverwaltung
    - Wie kann garantiert werden, dass ein historischer Stand eines Software-Projektes in allen Kopien des Repositories garantiert unveränderbar (inklusive Historie) ist?
- Etablierte Lösung
  - Blockchain-Technologie
    - Basiert auf den Merkle-Trees
      - Jeder Information wird der Hashwert der Vorgänger-Information hinzugefügt und daraus ein Hashwert berechnet
    - Kollisionen von Hashwerten sind prinzipiell möglich, aber absurd unwahrscheinlich

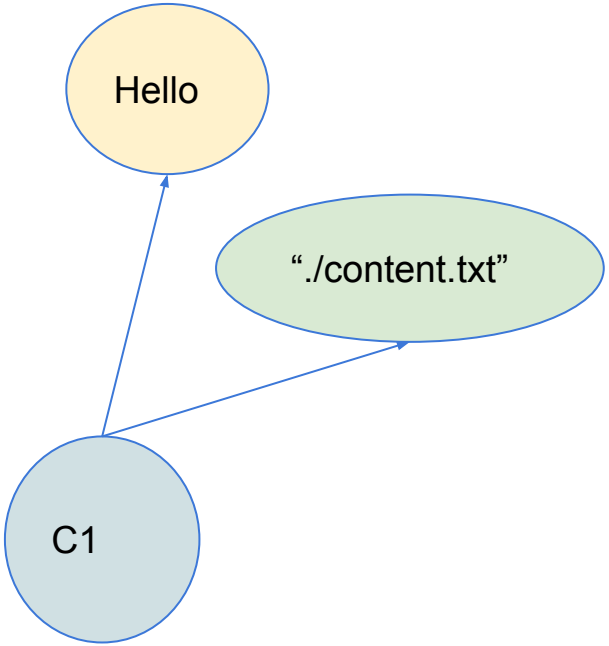
Git und Blockchain

- Git hat schon immer Blockchain-Technologie benutzt

Arbeitsweise von Git

- Alle Informationen werden in Git über berechnete Hashwerte identifizierbar gemacht
  - echo Hello > content.txt
  - git status
    - “Rote Datei”
  - git add content.txt
    - Im objects-Verzeichnis eine Datei e9/650…
  - git status
    - “Grüne Datei”
  - git commit -m “Commit Message”
  - git status
  - git log
    - Hashwert des Commits

Visualisierung (analog zum Speicher-Layout einer OOP-Sprache)



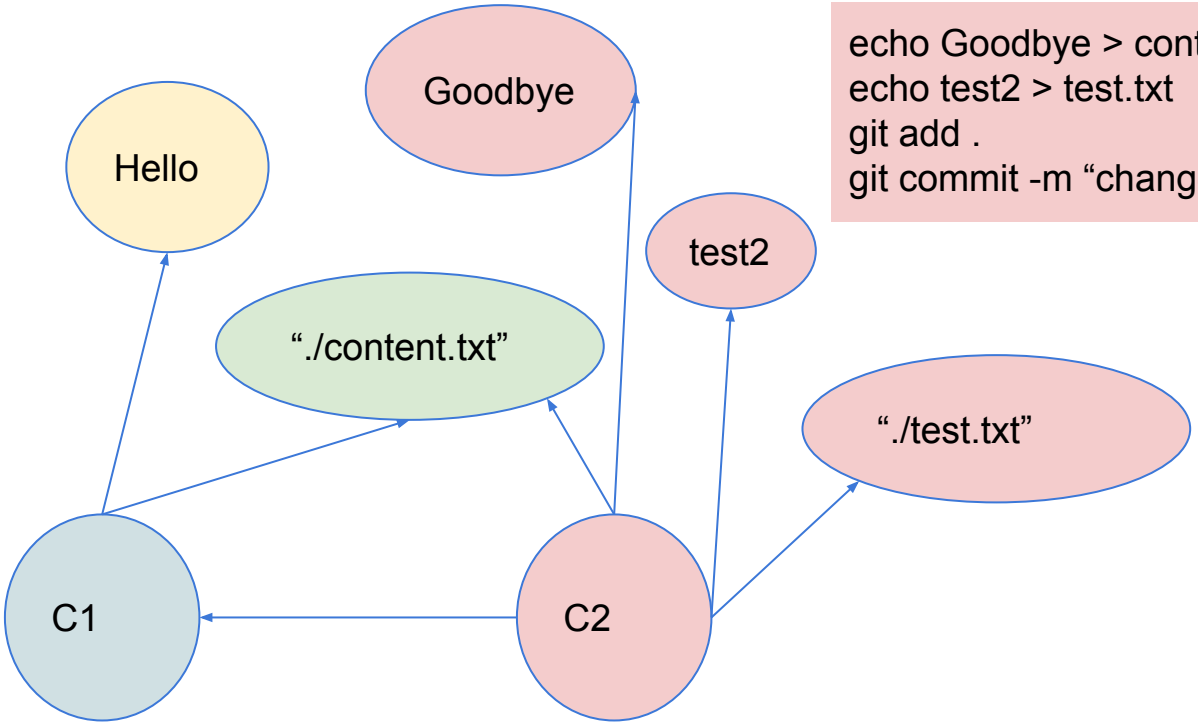
Content-Object
BLOB-Object

Tree-Object

Commit-Object

Anzeige: git log

Ein weiterer Commit




```
echo Goodbye > content.txt
echo test2 > test.txt
git add .
git commit -m "change"
```

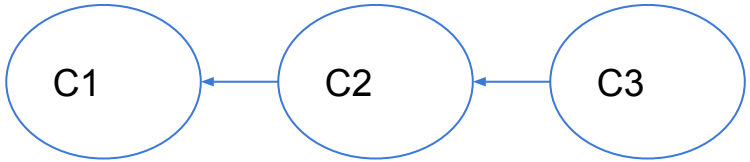
Content-Object
BLOB-Object

Tree-Object

Commit-Object

Anzeige: git log

Vereinfachte Visualisierung des Git Repository



Was macht der commit-Befehl?

- Ein commit erzeugt ein neues Commit-Objekt, das das Ausgangs-Commit-Objekt als Vorgänger enthält

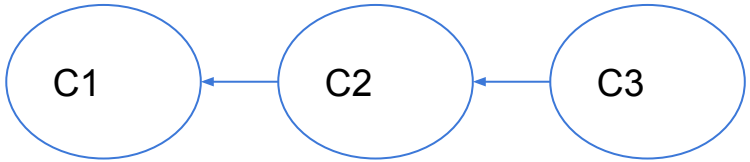
Ein paar “Details”

- git log
  - Ausgabe der Commits in einer Langform
- git log --oneline --graph --decorate --all
  - git config --global alias.pl “log --oneline --graph --decorate --all”
- git fsck --unreachable --no-reflogs
- Optionales Aufräumen (NICHT BESTANDTEIL DER NORMALEN ARBEIT)
  - git reflog expire --expire-unreachable=now
  - git gc --prune=now

Synchronisation des Workspaces mit einem Stand = Commit-Objekt

- git checkout <hash>
  - Empfehlung Sawitzki
    - “checkout nur bei unauffälligem Status”
      - “Nichts rotes, nichts grünes”
    - Falls Nein:
      - git add .
        - git commit -m “...”
        - git stash
          - Stashes sind nicht auf einen Server übertragbar
          - Verweis auf die git.pdf bzw. Online-Doku

git checkout <HASH>

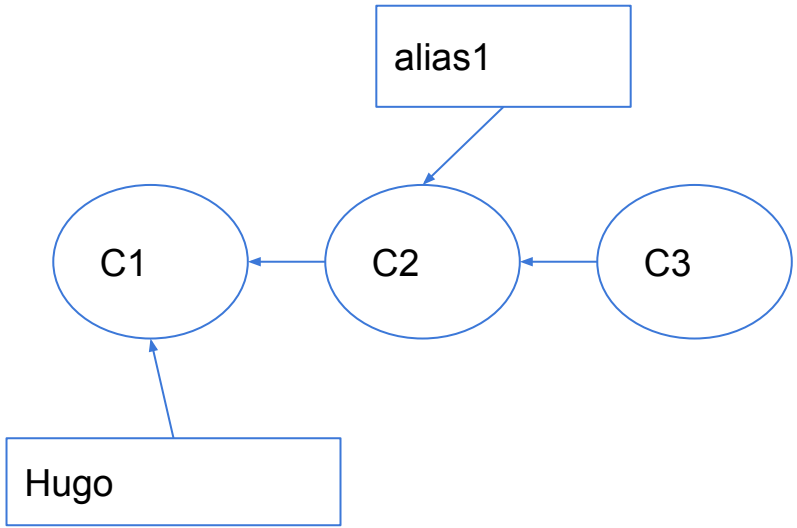


So arbeitet git intern immer

Ein Git-Anwender kann auch mit diesen Hash-Werten arbeiten -> “Nerd-Modus”

Alias-Namen auf Hash-Werte

- Statt langer Hashwerte können sprechende Alias-Namen benutzt werden



git checkout <alias>

Sinnvolle Kategorien eines Alias-Namens

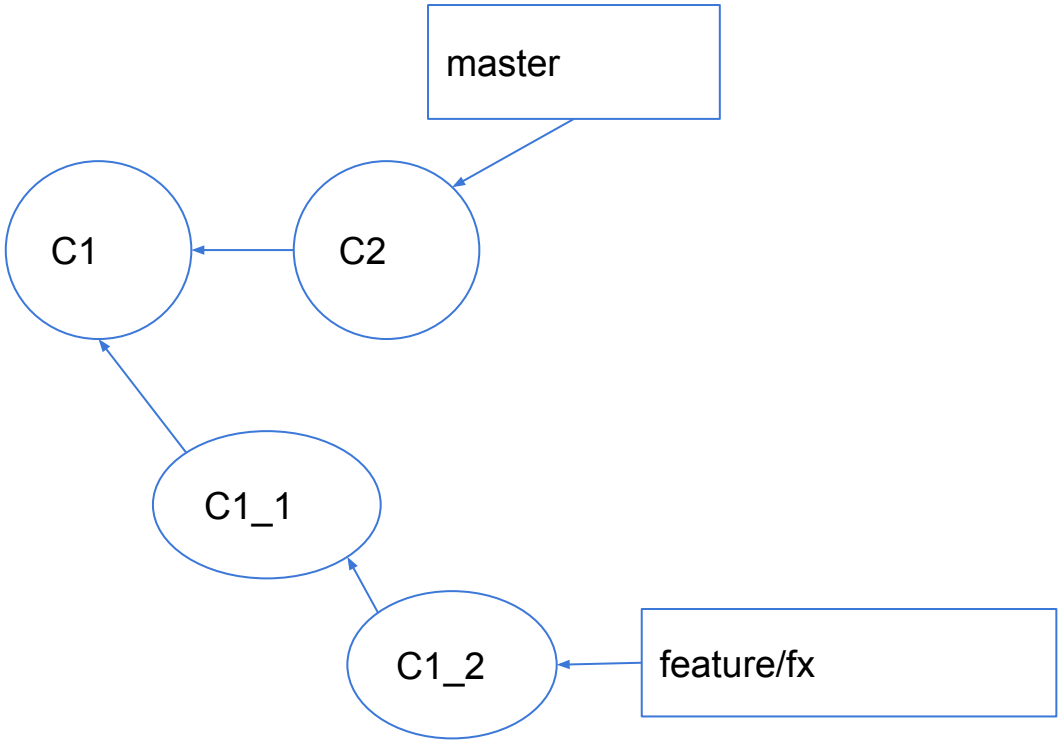
- Ein Commit-Objekt definiert einen fixen Stand
  - Beispiele
    - Release
      - v1.0
    - Milestone, Build-Nummer
    - “Heute Morgen”
  - Umsetzung mit git
    - Git Tags
      - git tag <name>
      - git tag --list
      - git tag -d <name>

Sinnvolle Kategorien eines Alias-Namens

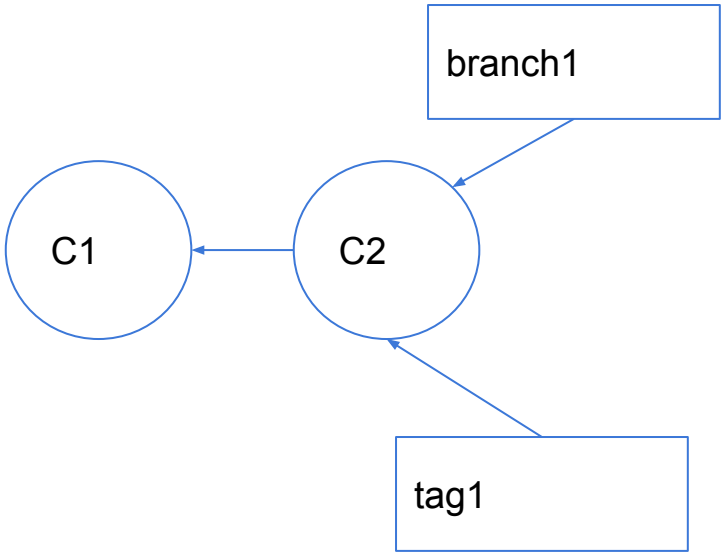
- Ein Commit-Objekt definiert einen aktuell durchgeführte Aktion
  - Beispiele
    - Entwicklung eines neuen Features
      - feature/webfrontend
    - Bugfix, Ticket-Nummers
    - “Heute Morgen”
  - Umsetzung mit git
    - Git Branches
      - git branch <name>
      - git branch --list
      - git branch -d <name>

WICHTIG

- Tags und Branches sind in Git absolut trivial
- Es sind und bleiben Alias-Namen

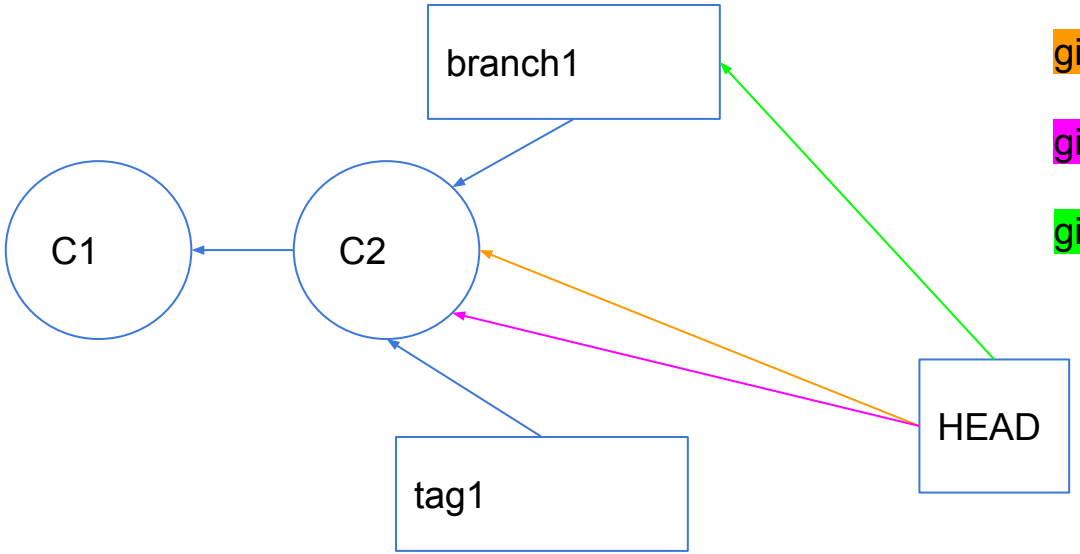

master
C1
C2
C1_1
C1_2
feature/fx

HEAD



HEAD ist ein Alias-Name, der im Geflecht der Commit-Objekte die aktuelle Position referenziert

Setzen des HEAD = git checkout



git checkout C2

git checkout tag1

git checkout branch1

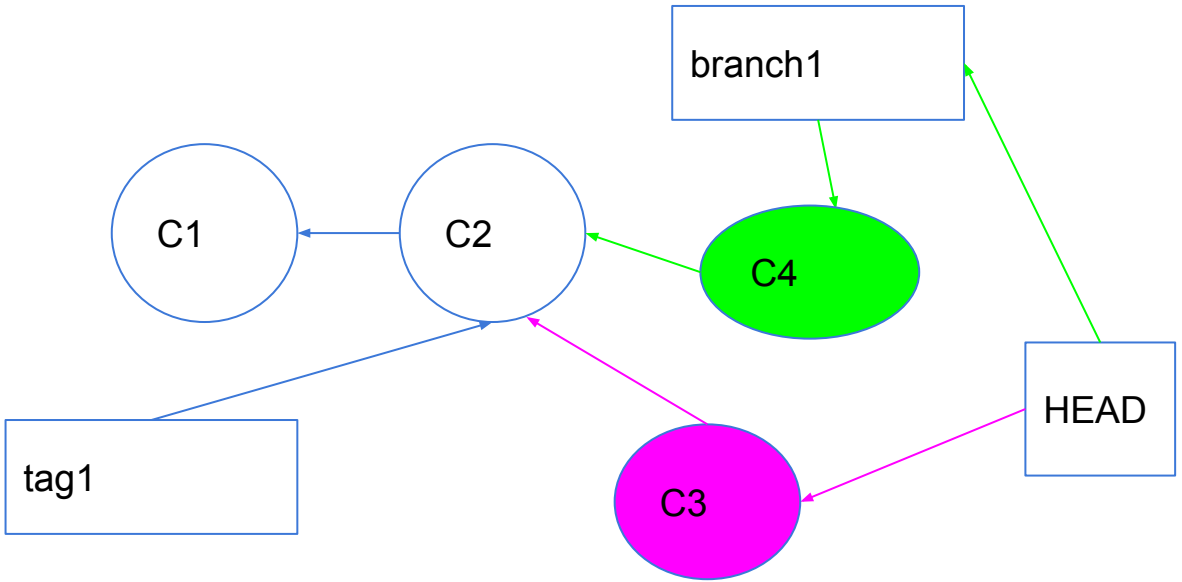
Detached HEAD

dann ist der Status auffällig

Attached HEAD

dann ist der Status unauffällig

git commit revisited




```
echo …
git add .
git commit -m “”
```

```
git checkout branch1
echo …
git add .
git commit -m “...”
```

Detached HEAD

Attached HEAD

Zusammenführen von Ständen

Ausgangssituation


```
$ git pl
* 44253d7 (feature2_part1) change content-feature2, part1
| * 05a2a0c (feature2_part2) change content-feature2, part2
|/
* c037e5a (feature2) add content-feature2
| * cdc92bc (feature1) add content-feature1
|/
* 3b74572 (HEAD -> master) change content
* f34626d add content
* f4aaddf setup project
```