



JAVACREAM

*Training
Consulting
Projectmanagement*

GIT

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



Ausgangssituation

- Ein Satz von Dateien repräsentiert einen Stand eines Projekts
 - angereichert mit Meta-Informationen
 - “Wer hat wann warum welche Änderungen gemacht”
- Parallele Fortentwicklung von Ständen
- Konsistente Zusammenführen (“Mergen”) von parallel entwickelten Ständen
- Zentrale Ablage aller Informationen
 - Authentifizierung und Autorisierung
- Verfahren und Methoden zur Zusammenarbeit im Team
- Unterstützung durch etablierte Verfahren zum effizienten Arbeiten

- Ein Satz von Dateien repräsentiert einen Stand eines Projekts
 - angereichert mit Meta-Informationen
 - “Wer hat wann warum welche Änderungen gemacht”
- Parallele Fortentwicklung von Ständen
- Konsistente Zusammenführen (“Mergen”) von parallel entwickelten Ständen
- Zentrale Ablage aller Informationen
 - Authentifizierung und Autorisierung
- Verfahren und Methoden zur Zusammenarbeit im Team
- Unterstützung durch etablierte Verfahren zum effizienten Arbeiten

- Ein Satz von Dateien repräsentiert einen Stand eines Projekts
 - angereichert mit Meta-Informationen
 - “Wer hat wann warum welche Änderungen gemacht”
- Parallele Fortentwicklung von Ständen
- Konsistente Zusammenführen (“Mergen”) von parallel entwickelten Ständen
- Zentrale Ablage aller Informationen
 - Authentifizierung und Autorisierung
- Verfahren und Methoden zur Zusammenarbeit im Team
- Unterstützung durch etablierte Verfahren zum effizienten Arbeiten

- Git Server sind Produkte von Herstellern außerhalb der Git-Community
 - Proprietär
 - Aufgrund der klaren Problemstellung sind aber die Produkte sehr ähnlich
- Übersicht
 - **GitHub Enterprise** (Microsoft)
 - GitLab (gitlab.com)
 - BitBucket (Atlassian)



- Ein Satz von Dateien repräsentiert einen Stand eines Projekts
 - angereichert mit Meta-Informationen
 - “Wer hat wann warum welche Änderungen gemacht”
- Parallele Fortentwicklung von Ständen
- Konsistente Zusammenführen (“Mergen”) von parallel entwickelten Ständen
- Zentrale Ablage aller Informationen
 - Authentifizierung und Autorisierung
- Verfahren und Methoden zur Zusammenarbeit im Team
- Unterstützung durch etablierte Verfahren zum effizienten Arbeiten



- Workflows zum effizienten Arbeiten mit Git
- Beispiele
 - Git Flow (Atlassian)
 - GitHub Flow (GitHub-Community)
- Hinweise
 - Sowohl Atlassian Git Flow als auch GitHub Flow sind Produkt-unabhängig
 - Git Flows sind immer als Templates zu verstehen, die im konkreten Einsatz an ein Projekt / ein Unternehmen angepasst werden

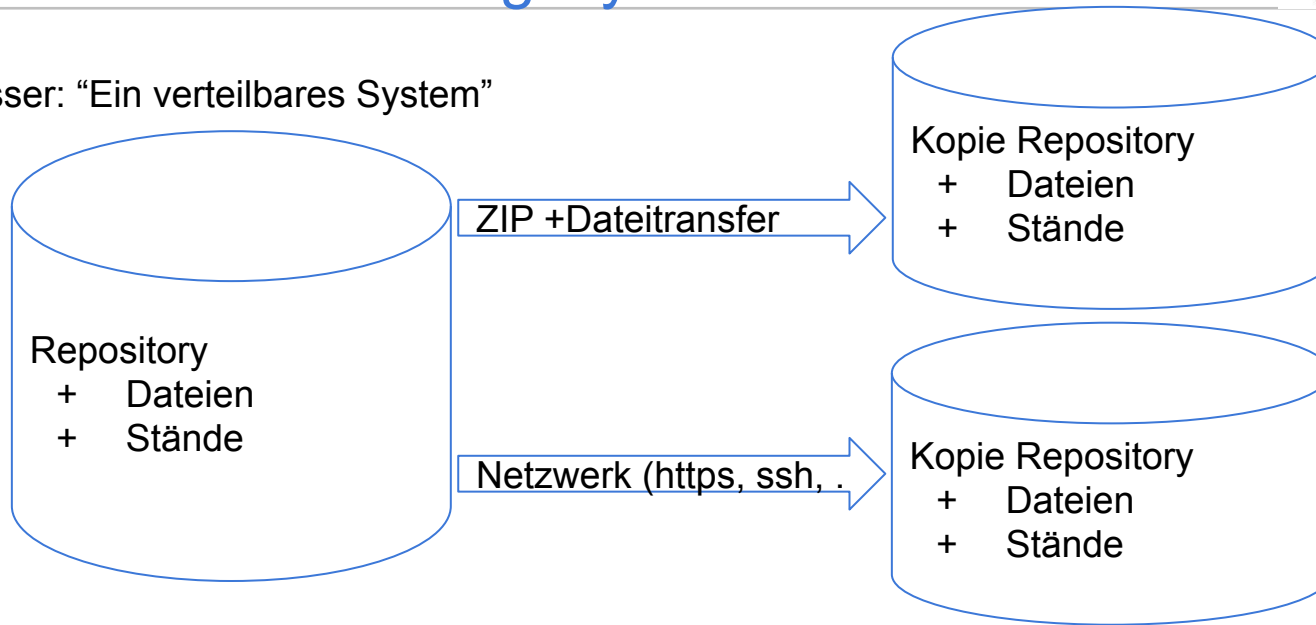
- Ein Satz von Dateien repräsentiert einen Stand eines Projekts
 - angereichert mit Meta-Informationen
 - “Wer hat wann warum welche Änderungen gemacht”
- Parallele Fortentwicklung von Ständen
- Konsistente Zusammenführen (“Mergen”) von parallel entwickelten Ständen
- Zentrale Ablage aller Informationen
 - Authentifizierung und Autorisierung
- Verfahren und Methoden zur Zusammenarbeit im Team
- Unterstützung durch etablierte Verfahren zum effizienten Arbeiten

Tag 1 + Tag 2 erste
Session

Git ist ein “verteiltes Versionsverwaltungssystem”



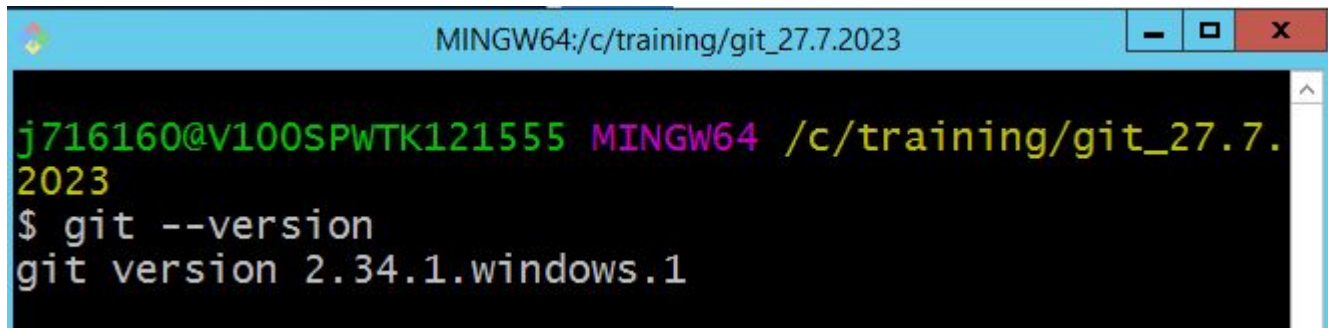
Besser: “Ein verteilbares System”



Unbedingt notwendig ist eine Fälschungs-sichere Konsistenz der verteilten Informationen
Lösung wird durch Merkle-Trees erreicht (Informationen bestimmen einen Hashwert, Der Hash der Vorgänger-Information wird dabei mitberücksichtigt) -> Blockchain
GIT benutzt / arbeitet mit Blockchain-Technologie

First Contact

- Kommandos über ein Terminal



```
MINGW64:/c/training/git_27.7.2023

j716160@V100SPWTK121555 MINGW64 /c/training/git_27.7.2023
$ git --version
git version 2.34.1.windows.1
```

- Installation native oder “Portable Git”
 - Git ist kein Hintergrund-Dienst
 - Die Git-Kommandos stellen das komplette Versionsverwaltungssystem bereit

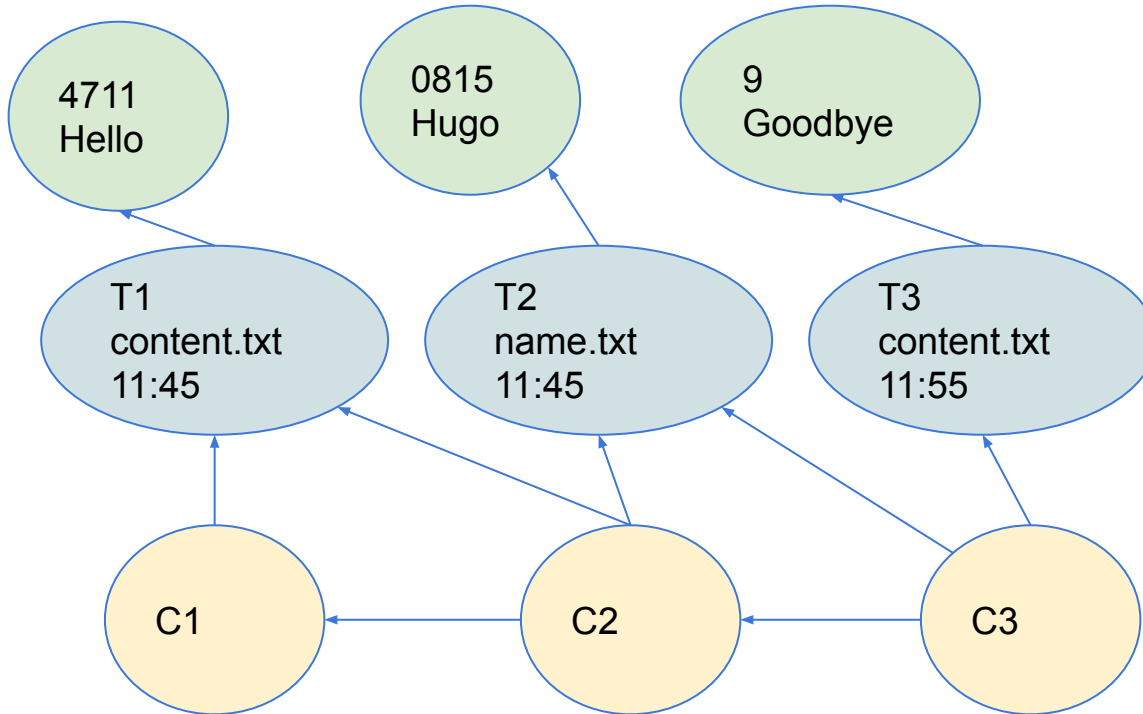
- Konfiguration von Benutzer-Name und eMail-Adresse
 - `git config --global user.name`
 - `git config --global user.email`
- Hinweise
 - Bei Ihnen bereits gesetzt
 - Check: `git config --get user.name`
 - `--global`
 - Global heißt: Für den Benutzer, `.gitconfig` im UserHome-Directory
 - `--local` (pro Repo) oder `--system` (Rechner-übergreifend)
 - `-gitconfig` ist eine strukturierte Text-Datei

- **Untypisch: Initialisieren eines lokalen, leeren Repositories**
 - Im Seminar erst einmal die einzige Möglichkeit
 - In der Praxis würden Sie ein Repo auf GitHub anlegen und clonen -> später
- **mkdir training, cd training**
 - training ist ein ganz normales Verzeichnis
- **git init**
 - Erzeugt in training das Unterverzeichnis .git
 - Wording
 - -git -> Git Repository
 - training ist immer noch “normal”, aber ich nenne es ein Git-Projektverzeichnis
 - Alle Dateien außerhalb von -git sind Bestandteil des Workspaces
 - Check
 - Im Prompt steht “(master)”
 - git status erzeugt nur unauffällige Ausgaben
- Hinweis: Ein Löschen des Verzeichnisses ,git zerstört das Repository unwiderruflich

- Erstellen einer Datei im Workspace
 - `echo Hello > content.txt`
- `git status` zeigt eine Inkonsistenz an: Es existiert eine Datei im Workspace, die dem Repository unbekannt ist
 - Datei ist “rot”
 - Kein Fehler!
- `git add content.txt`
 - Parameter: Liste von Dateien mit Platzhaltern
- `git status` zeigt eine Inkonsistenz an: Das Repository enthält in einer “Staging Area” eine Datei, die noch keinem Stand zugeordnet ist
 - Datei ist “grün”
 - Es ist nicht alles “OK”
-

- Definition eines Standes erfordert das Erfassen der Meta-Informationen
 - Wer (user.name) hat wann (timestamp) warum (?) welche (Informationen in der Staging Area) gemacht
 - `git commit -m "Beschreibung: warum? - > Commit Message"`
- `git status` ist unauffällig
- `git log`
 - Ausgabe der letzten Meta-Informationen eines Standes

- Normales Arbeiten im Workspace
- Git
 - `git add .`
 - `git commit -m "neuer Stand"`



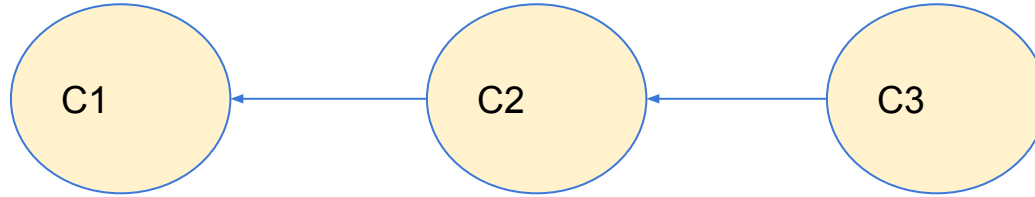
Content oder BLOB
Objekt

Tree Objekt

Commit Objekt



- `git add`
 - erzeugt ausschließlich Content-Objekte
- `git commit`
 - erzeugt die Tree-Objekte
 - Erzeugt ein neues Commit-Objekt mit Verweis auf den Vorgänger
 - Blockchain



Commit Objekt

git commit

- + Erzeugt ein neues Commit-Objekt mit Verweis auf den Vorgänger

- `git checkout <hash>`
 - Damit wird der Workspace mit einem Stand synchronisiert = überschrieben
- Empfehlung Sawitzki
 - “Checkout nur bei unauffälligen Status”
 - Falls auffällig
 - `git add .`
 - `git commit -m “...”`
 - `git stash`
 - Im Wesentlichen ein lokaler Backup der Staging-Area
 - Bitte aber nur im “Notfall”
 - Details -> Dokumentation / Online

- Nachvollziehen
- Ist ein Commit auch auf “alten” Ständen möglich?
- Wie würden Sie das im Graphen der Commit-Objekte visualisieren?
- Vorsicht:
 - Behalten Sie die Hash-Werte in der Historie, ein git log zeigt nicht immer alle Hashes an