Einführung in die Versionsverwaltung mit Git

SS 2015

Version 1.0

Prof. Dr. Michael Munz
Fakultät für Mechatronik und Medizintechnik
Hochschule Ulm

Technik
Informatik & Medien
Hochschule Ulm
University of
Applied Sciences

Vorteile Versionsverwaltung

- Gleichzeitiges Arbeiten an gemeinsamer Codebasis
  - Dateien können sogar gleichzeitig bearbeitet werden
  - Automatisches Zusammenführen der Dateien ohne Kopieren von Hand
- Es kann jederzeit zu einer früheren Version zurück gewechselt werden
- Änderungen können verfolgt werden (wer hat was geändert)
  - Realisierung von Rückverfolgbarkeits-Forderungen
  - Erleichtert Fehlersuche
  - Eingrenzung von Fehlern möglich (wann hat es zum letzten Mal funktioniert?)
  - Auswertungen: welche Module, Klassen,… werden sehr häufig geändert, welche sind stabil?
- Backup-Funktionalität
- Es können verschiedene Varianten (Branches) verwaltet werde (z.B. „Feature Branches“)
- Versionsstände können markiert und damit eindeutig identifiziert werden (Tags)

Typische Use-Cases

- Auschecken des aktuellen Software-Stands (Änderungen von anderen Entwicklern)
- Vergleichen von Versionsständen oder einzelnen Dateien (Was wurde geändert?)
- Erstellen von Branches und Zurückführen von Branches auf den Hauptentwicklungszweig
- Zurücksetzen von Änderungen

Übersicht über Versionsverwaltungssysteme

Kommerziell:

- Microsoft Visual SourceSafe (VSS) (wird nicht mehr weiterentwickelt)

  →Nachfolger: Microsoft Team Foundation Server (Express-Version für max. 5 Personen kostenlos)

Open Source:

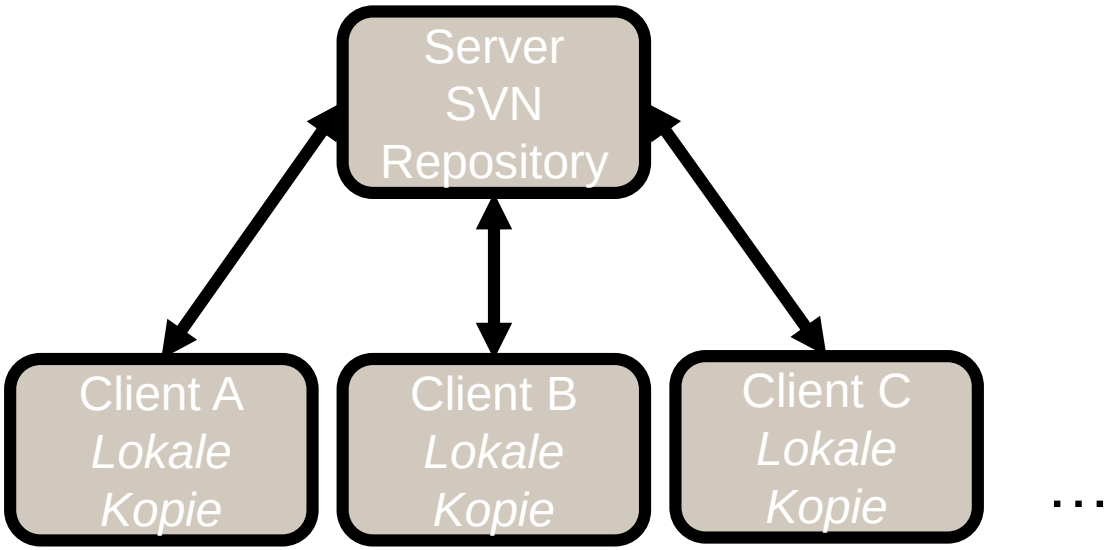
- Concurrent Versioning System (CVS) (wird nicht mehr weiterentwickelt)

  →Nachfolger Apache Subversion (SVN)
- Git (http://git-scm.com/)

Allgemeines

Apache Subversion (SVN):

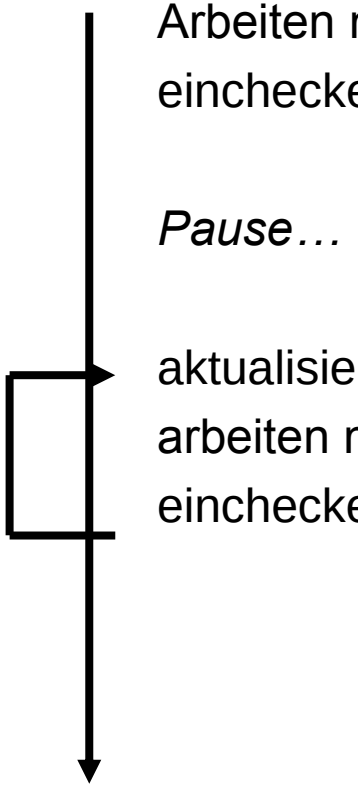
- Freie Software zur Versionsverwaltung (Apache License 2.0)
- Zentrales Repository (Server)
- Nachfolger von CSV (Concurrent Versioning System)



Arbeiten mit SVN

Client mit grafischer Benutzeroberfläche unter Windows: Tortoise SVN

Standard-Arbeitsablauf:



Initial: Repository auschecken (svn checkout) …

Arbeiten mit Dateien…

einchecken (svn commit) …

Pause…

aktualisieren des lokalen Repositories (svn update)…

arbeiten mit Dateien…

einchecken (svn commit)…

Das Versionsverwaltungssystem Git

- Lizenz: GNU GPLv2
- Entwickelt für die Linux-Kernelentwicklung (Linus Torwalds)
- Bedeutung des Namens: „Blödmann“

“I’m an egotistical bastard, and I name all my projects after myself. First ‘Linux’, now ‘Git’.”
– Linus Torvalds

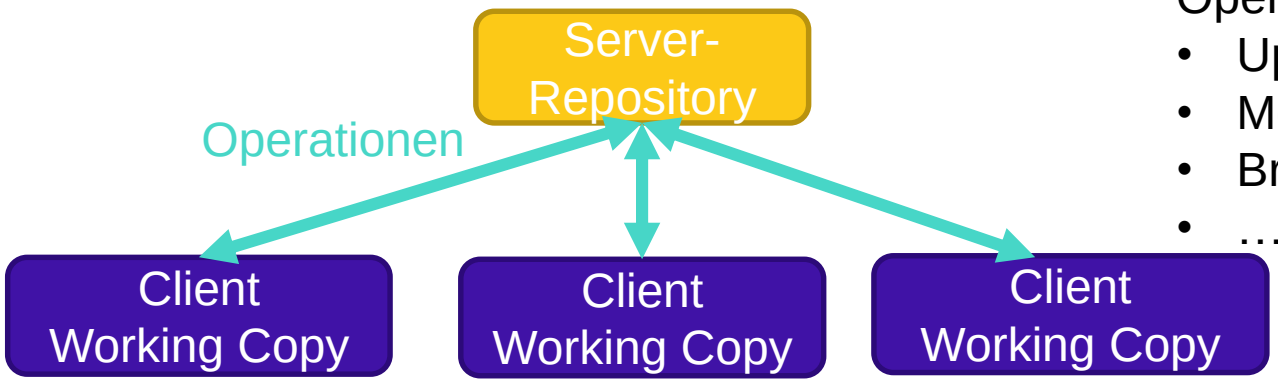
- Modernstes Versionsverwaltungssystem
  - Verteile Architektur
  - Hohe Sicherheit gegenüber unbeabsichtige wie beabsichtige Verfälschung
  - Hohe Effizienz
  - Keine Locks, gleichzeitige Bearbeitung von Quellcode, nachträgliches Zusammenführen (merging)

  → Benötigt keinen **Server**!

Git - Eigenschaften

Git besitzt eine dezentrale Architektur

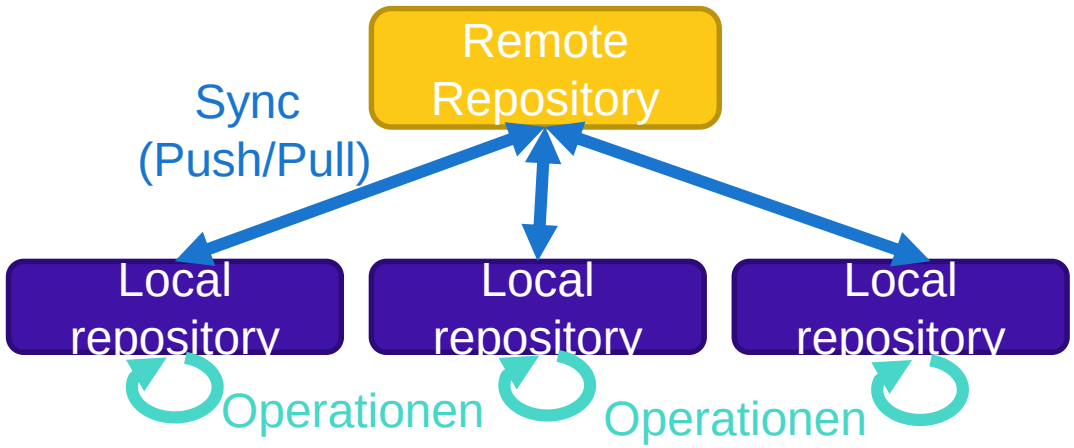
- Herkömmliche Client-Server-Architektur

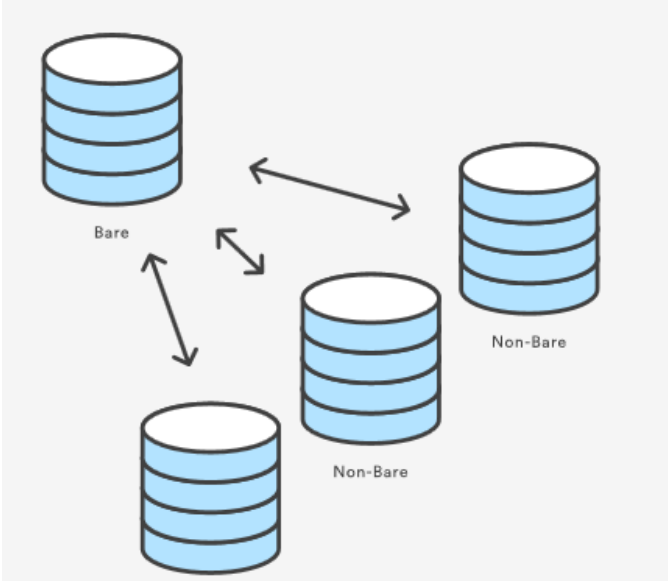


Operationen können sein:

- Update/Commit
- Merge
- Branch
- …

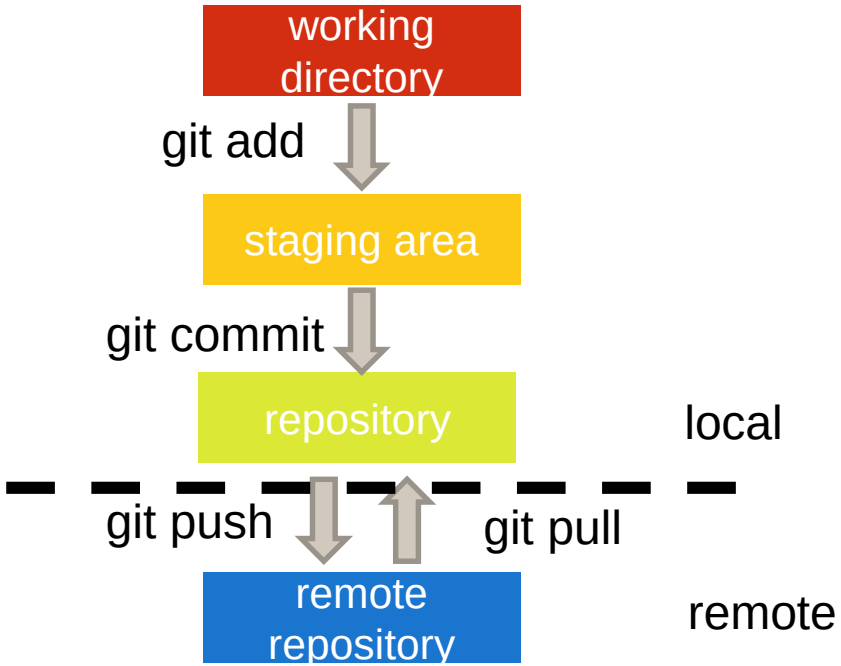
- Dezentrale Architektur → Kein Server





Verwaltungs-Ebenen in Git

Es gibt bei Git mehrere Ebenen:



git add: Datei oder Änderung in einer Datei wird zur Staging Area hinzugefügt

git commit: Änderungen in der Staging Area werden in das Repository übertragen

git push: Lokales Repository wird an das remote Repository übertragen

git pull: Änderungen des remote Repositories werden in das lokale Repository übertragen

Änderungsverwaltung

- Git erlaubt es, kleinste Änderungen (auch einzelne Zeilen) separat im Versionsverwaltungssystem zu übermitteln
- Jeder (!) Commit sollte eine Commit-Message besitzen
  - Dienen dazu, den Sinn der Änderungen nachvollziehen zu können
  - Änderungen können wieder gefunden werden
- Jede Änderung (Commit) besitzt in Git eine eindeutige ID (SHA-1 Hash), 160 bit
  - Hash wird meist als Hexadezimal-Zeichenkette dargestellt (40 Zeichen)
  - Meistens sind innerhalb eines Repositories die ersten 6 Zeichen bereits eindeutig!
  - Jede Änderung führt damit zu einer neuen Revision (Versionsstand)
  - Eine Revision gilt immer das gesamte Repository (nicht nur für die einzelne Datei!)
- In Git lassen sich nur Dateien (und deren Pfade) verwalten, keine Verzeichnisse direkt (d.h. keine leeren Verzeichnisse)

Dateien ignorieren

- In Git können bestimmte Dateien ignoriert werden
  - Abhängig vom Dateinamen, Teile des Dateinamens (wildcards) oder Dateierweiterung
  - Diese werden in die Datei .gitignore eingetragen (wird auch im Repository verwaltet)

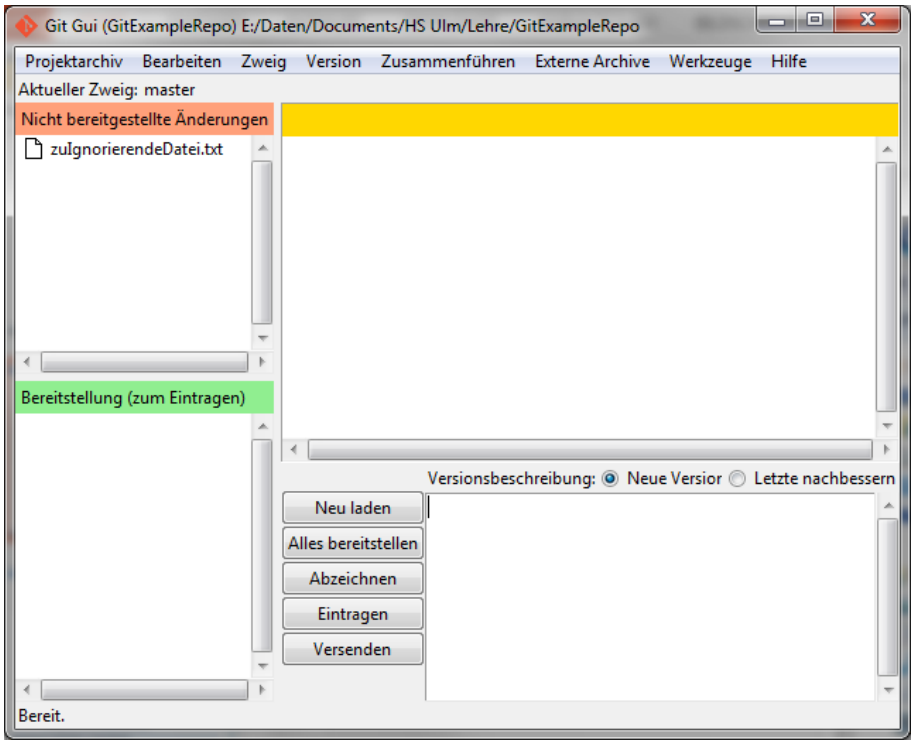
Tipp: Am besten immer folgende Dateien/Typen/Verzeichnisse ignorieren:

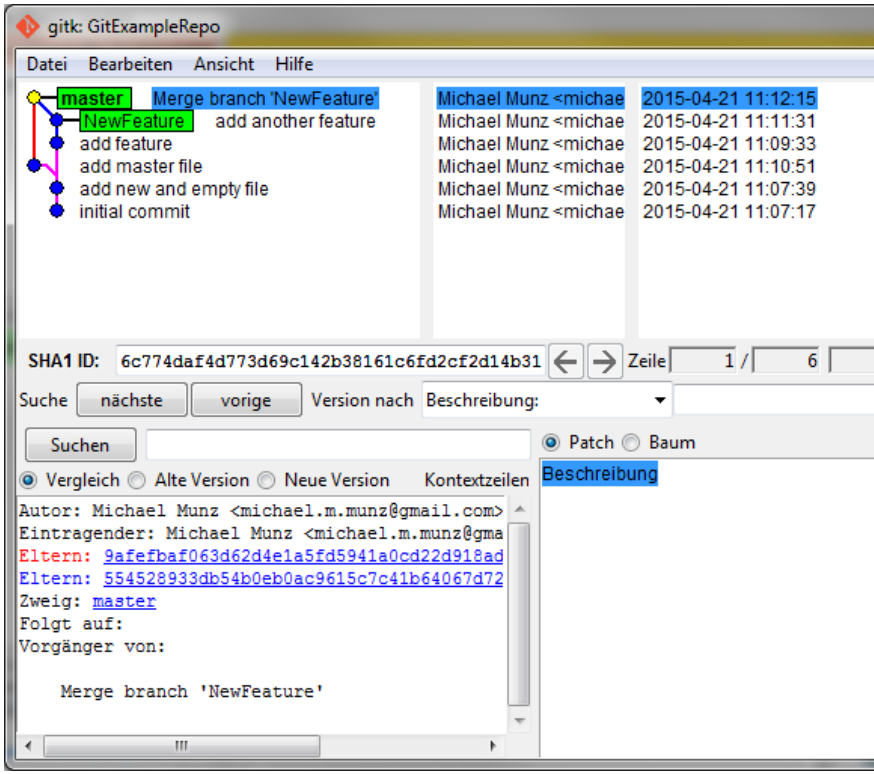
- Compilate (.exe, .jar, .class etc.)
- Bin-Verzeichnisse
- Lokale Einstellungen der Entwicklungsumgebungen (z.B. Visual Studio .suo-Dateien)
- Zwischenergebnisse von Entwicklungsumgebungen aller Art

GUI-Clients

- Für git existieren einige kostenlose grafische Benutzeroberflächen, die die Arbeit erheblich vereinfachen
- Neben den grafischen Oberflächen sind Plugins für viele Entwicklungsumgebungen (Visual Studio, Eclipse, …) verfügbar

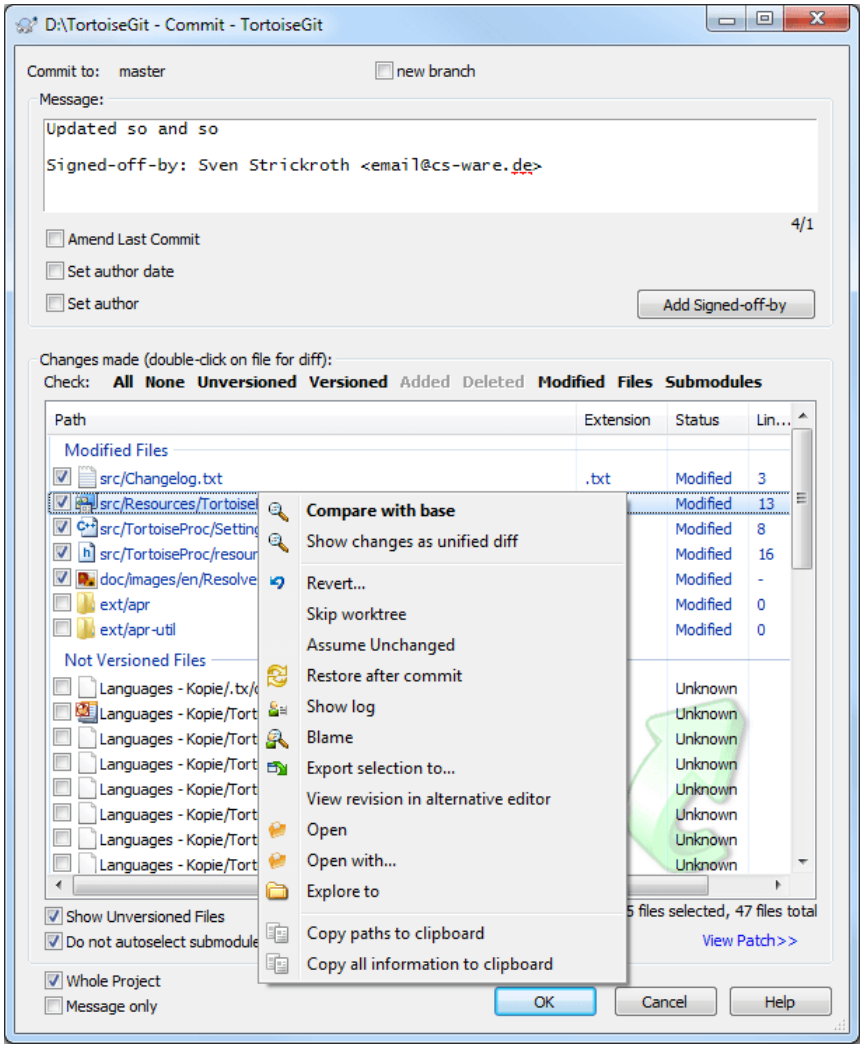
Mit git direkt mitgeliefert: **git gui** und **gitk**





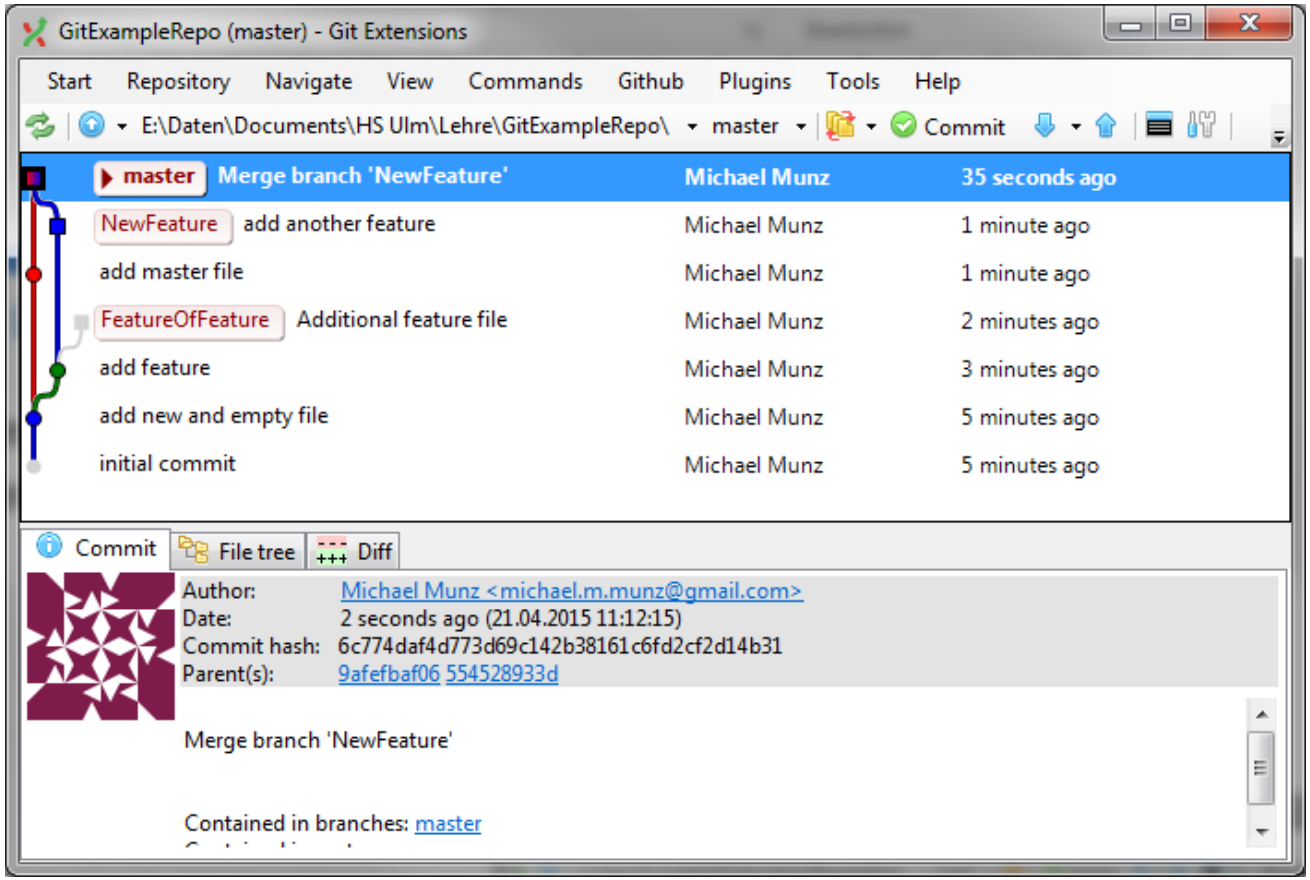
GUI-Clients (2)

TortoiseGit (Windows): Kontext-Integration im Explorer + Oberfläche



GUI-Clients (3)

- **GitExtensions** (Windows): Kontext-Integration im Explorer + Oberfläche



Befehle für einen gewöhnlichen Arbeitsablauf

- Clonen eines bestehenden Remote-Repositories:
  `git clone <URL>`

- Änderungen im Verzeichnis (= neue Datei, Datei gelöscht, etc.) oder an einer Datei (= Inhalt geändert) für einen Commit vorbereiten (→ in die Staging Area legen):
  `git add <dateiname>`

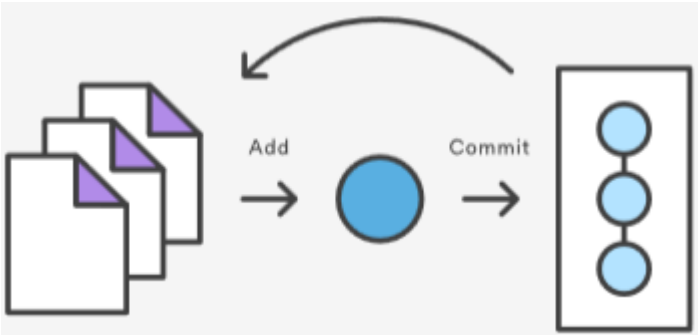
- Änderungen in der Staging Area in das Repository übertragen:
  `git commit –m <“Commit-Message“>`

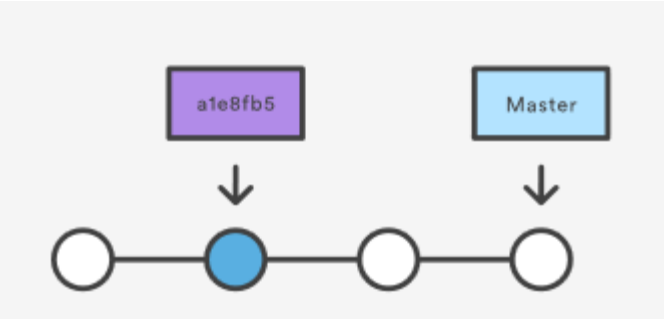
- Änderungen an ein Remote-Repository übertragen:
  `git push <repository> <branch>`
  (Meist: `git push origin master`)

- Änderungen von einem Remote-Repository erhalten:
  `git pull`
  (führt gleichzeitig git fetch (Holen der Änderungen) und git merge durch; kann auch nacheinander ausgeführt werden)

Befehle für einen gewöhnlichen Arbeitsablauf

- Änderungen rückgängig machen
  1. Auf den letzten Stand von HEAD (lokal) zurücksetzen (alle bereits committeten Änderungen bleiben erhalten):
     ```
     git checkout -- <dateiname>
     ```
  2. Auf den Stand des Remote Repositories zurücksetzen:
     ```
     git fetch origin
     git reset –hard origin/master
     ```
  3. Einen commit rückgängig machen:
     ```
     git revert <commit-id>
     ```
  4. Alles oder einzelne Datei auf einen bestimmten Versionsstand zurücksetzen (→ detached head):
     ```
     git checkout <commit-id> [<file>]
     ```





Zusätzliche Befehle

- Neues Remote hinzufügen
  ```
  git remote add <name> <branch-name>
  ```
  Falls z.B. das Repository nicht von remote geclont wurde:
  ```
  git remote add origin master
  ```

- Einen neuen Branch erstellen und zu diesem wechseln:
  ```
  git checkout –b <branchname>
  ```

- Zu einem bestehenden Branch wechseln:
  ```
  git checkout <branchname>
  ```

- Einen bestehenden Branch wieder löschen:
  ```
  git branch –d <branchname>
  ```

- Branch an das Repository übertragen:
  ```
  git push origin <branchname>
  ```

Terminologie bei Git

Begriff	Bedeutung
Branch	Ein Branch ist ein benannter Pointer zu einem Commit. Durch das Auswählen eines Branches wird dieser ausgecheckt. Neue Branch können von einem existierenenden Branch angelegt werden, was die Änderung des Codes unabhängig von anderen Branches erlaubt. Ein Branch ist immer der default (normalerweise master genannt)
Commit	Wenn ein Commit von Änderungen im Repository durchgeführt wird, wird ein Commit Objekt erzeugt. Dieses Objekt beschreibt eindeutig eine neue Revision des gesamten Repository-Inhalts. Diese Revision kann später wiedergefunden werden (Änderungsvergleich, Zurücksetzen, etc.)
HEAD	Head ist eine symbolische Referenz, die normalerweise auf den aktuell ausgecheckten Branch zeigt. Manchmal zeigt HEAD direkt auf ein Commit-Objekt. Dies ist dann der detached HEAD mode. Dann führen Commits nicht zu einer Änderung eines Branches. Wenn Branches gewechselt werden, zeigt der HEAD auf den Branch, der wiederum auf den letzten Commit zeigt. Wenn ein spezifischer Commit (Revision) ausgecheckt wird, dann zeigt HEAD direkt auf diesen Commit.
Index	Index ist ein anderes Wort für Staging Area.
Repository	Ein Repository enthält die gesamte Versionshistory. Es gibt bare Repositories (ohne lokale Dateien) und personal Repositories (inkl. Working Copy). Bare Repositories erlauben nur das Übertragen (push/pull) von Versionsänderungen und werden gewöhnlich als Remote Repositories eingesetzt.
Revision	Repräsentiert eine Version des gesamten Source-Codes. Wird in Git durch Commit Objekte (Commits) implementiert. Diese werden durch SHA-1-Hashes identifiziert.
Staging area	Staging area ist der Ort, in dem Änderungen gesammelt werden bevor sie per Commit in das Repository (lokal) übertragen werden.
Tag	Mit Tags können Commits eindeutig gekennzeichnet werden. Entspricht einem benannten Pointer zu einem Commit. Tags können neben dem Namen auch Beschreibungen beinhalten.
URL	Eine URL bezeichnet den Ort eines Repositories. Es wird zwischen Fetch-URL (Holen von Änderungen) und Push-URL (Übertragen von Änderungen an Remote Repository) unterschieden.
Working tree	Enthält alle Arbeitsdateien (Source-Code, etc.) des Respositories. Mit diesen wird gearbeitet. Änderungen werden dann an das Repository übertragen.

Tipps

- **Konflikte** sollten durch Absprachen (Aufteilung der Methoden und Klassen oder Dateien) **möglichst vermieden** werden
  → spart Arbeitsaufwand
  → bei der Konfliktlösung (mergen) können immer Fehler passieren
- Keine Kompilate oder Objektdateien einchecken (Verzeichnisse „build“, „dist“, etc.)
  → gibt immer Modifikationshinweise
  → Verzeichnise oder Dateityp-Filter in git-ignore-Datei eintragen (werden vom System ignoriert)
- Nur compilierenden Code pushen
  → vermeidet Ärger
- Kommentare bei Commits verwenden, häufig commiten (atomare Commits) und Commits nach Inhalt trennen
  → Nachvollziehbarkeit (wer hat wann was geändert)

Tutorials und Downloads:

http://git-scm.com/book/de/v1

http://www.vogella.com/tutorials/Git

http://rogerdudler.github.io/git-guide/index.de.html

https://www.atlassian.com/git/tutorials/

Git:

http://git-scm.com/

GitPortable:

https://github.com/sheabunge/GitPortable/releases/download/v1.9.5-devtest.1/GitPortable_1.9.5_Development_Test_1_online.paf.exe

GitExtensions GUI:

https://code.google.com/p/gitextensions/