Git Grundlagen – Team-Handout

Was ist Git und warum sollte man es verwenden?

Was ist Git?
Git ist ein Tool zur Versionskontrolle von Softwareprojekten. Es ermöglicht, Änderungen an diesen Projekten zu verfolgen und bietet den Entwickler*innen die Möglichkeit, ihre Projekte zu verwalten und an diesen gemeinsam zu arbeiten.

Warum sollte man Git verwenden?

- Nachvollziehbare Versionshistorie und einfache Rückverfolgung von Änderungen,
- Verteiltes Arbeiten durch vollständige lokale Kopie des Repositories,
- Branching und Merging erleichtern parallele Entwicklungsprozesse,
- Kryptografische Hashes (z. B. SHA-1, SHA-256) sorgen für die Integrität der Historie,
- Schnelle und effiziente Arbeitsweise, auch bei großen Projekten uvm.

***Quelle**: GitLab – Was ist Git? Der ultimative Leitfaden*

Grundlegende Git-Befehle

Start

`git init` : Initialisiert ein neues, leeres Git-Repository im aktuellen Verzeichnis. Ein **.git-Ordner** wird erstellt, worin alle Informationen zur Versionskontrolle abgelegt werden.
`git clone <URL>` : Repository von URL klonen → Kopiert ein bestehendes Remote-Repository (z. B. von GitHub) auf das eigene lokale System. Dabei wird ein Arbeitsordner erstell worin der Projektinhalt gespeichert wird sowie die Versionsinformationen.

Änderungen in Staging Area

`git status` : Zeigt den aktuellen Status `git add <Datei>` : eine oder mehrere Dateien werden zur Staging-Area hinzugefügt
`git add .` : fügt alle geänderten Dateien hinzu `git diff` : Vergleicht aktuellen Stand mit letzter Version - macht Änderungen der Dateien Zeilenweise sichtbar. (Nur außerhalb der Staging-Area)

Versionsänderungen speichern

`git commit -m "Nachricht"` : Speichert alle aktuell in der Staging-Area befindlichen Änderungen als neue Version (Commit) mit einer Nachricht. `git commit -am "Nachricht"` : Kombiniert add und commit, erfasst jedoch nur Änderungen an bereits versionierten Dateien. Neue Dateien werden nicht berücksichtigt.

Versionsänderungen abrufen oder veröffentlichen

`git pull` : Holt neue Änderungen vom Remote-Repository und integriert sie automatisch in den aktuellen Branch `git push` : Überträgt lokale Commits auf das Remote-Repository, um sie für andere verfügbar zu machen. `git fetch` : Lädt neue Daten vom Remote-Repository herunter, ohne sie automatisch zu integrieren.

Änderungen rückgängig machen

`git stash` : Speichert aktuelle Arbeitsänderungen temporär, um den Arbeitsordner zu leeren (KEIN Commit). `git reset` : Setzt den aktuellen Branch auf einen bestimmten Commit zurück. Modus: `--soft, --mixed, --hard` `git restore <Datei>` : Datei wird auf den Zustand des letzten Commits zurückgesetzt.

Branches verwalten

`git branch` : Listet alle lokalen Branches
`git branch -a` : Listet auch die remote Branches
`git checkout <Branch>` : Wechselt in einen anderen Branch und aktualisiert den Arbeitsstand entsprechend.
`git switch <Branch>` : Branch-Wechsel
`git merge <Branch>` : Änderungen in den aktuellen Branch zusammenführen.

Branches und ihre Nutzung, Umgang mit Merge-Konflikten

Branches ermöglichen paralleles Arbeiten, indem sie voneinander separate Entwicklungszweige schaffen. So können mehrere Funktionen unabhängig entwickelt und später zusammengeführt werden.

Beim Zusammenführen, also dem "Merge", kann es zu Konflikten kommen. Gerade dann, wenn in beiden Branches die gleichen Stellen einer Datei verändert wurden. Die Konflikte betreffen nur die Person, die den Merge durchführt. Der Rest des Teams bemerkt von dem Konflikt nichts. In diesem Fall kann Git nicht automatisch erkennen, welche Änderung genau übernommen werden soll und unterbricht deswegen den Merge-Prozess.

Grundlegend kann man zwei Arten von Merge-Konflikten definieren:

1. Git kann Merge nicht starten

   Ein Merge-Vorgang kann fehlschlagen, wenn sich im Arbeitsverzeichnis oder im Staging-Bereich nicht gespeicherte Änderungen befinden. Git verhindert in diesem Fall den Merge, um zu vermeiden, dass diese lokalen Anpassungen von den neuen Commits überschrieben werden.

   Dabei handelt es sich nicht um einen Konflikt mit anderen Branches, sondern um Konflikte mit eigenen, noch nicht gesicherten Änderungen. Um fortzufahren, sollte der lokale Zustand bereinigt oder gesichert werden.

   ```
   git stash     - Änderungen zwischenspeichern
   git commit    - Änderungen festschreiben
   git reset     - Änderungen verwerfen
   git checkout  - zu einem anderen Branch wechseln
   ```

Erst danach kann der Merge erfolgreich ausgeführt werden.

2. Während des Merge-Vorgangs tritt ein Fehler auf

   Tritt ein Fehler während eines Merge-Vorgangs auf, gibt es Konflikte zwischen dem aktuellen Branch und dem Branch, der gemergt wird. Diese Konflikte entstehen dann, wenn einer oder mehrere Personen denselben Codeabschnitt verändert haben.

   Git versucht hier dann, die Änderungen automatisch zusammenzuführen, kann dies jedoch nicht in allen Fällen. Die betroffenen Dateien müssen dann manuell überprüft und angepasst werden. In solchen Fällen zeigt Git eine entsprechende Fehlermeldung an, die auf den Konflikt hinweist und die betroffenen Dateien nennt.

Merge-Konflikte beheben

Ein Weg, einen Merge-Konflikt zu lösen, ist das manuelle Bearbeiten der betroffenen Datei. Man öffnet dazu die Datei in deinem Editor, entfernt die Konfliktmarker und übernimmt die gewünschten Änderungen.

```
git add <Dateiname>
git commit -m "Merge-Konflikt in <Dateiname> behoben"
```

Git erkennt, dass der Konflikt gelöst wurde, und erstellt dann automatisch einen Merge-Commit um den Vorgang abzuschließen.

Nützliche Befehle bei Merge-Konflikten:

```
git status          - zeigt Status der Dateien
git log             - zeigt Commit-Verlauf
git log --merge     - Protokoll-Liste wird erstellt, die zwischen mergenden
Branches für Konflikte sorgen.
git checkout        - wechselt zu einem anderen Branch oder verwirft
Änderungen
git reset           - setzt Änderungen zurück
git merge --abort   - Merge-Prozess verlassen, Branch auf Status vor dem
Merge zurücksetzen.
```

Git mit IntelliJ benutzen: Local Repository und Remote Repository

IntelliJ IDEA bietet eine integrierte Git-Unterstützung, wodurch sich alle grundlegenden Git-Funktionen direkt aus der Entwicklungsumgebung heraus nutzen lassen. So kann sowohl das lokale als auch das Remote Repository ohne die Kommandozeile verwaltet werden.

Wie in der Sektion „Grundlegende Git-Befehle“ beschrieben, wird beim Ausführen von `git init` im Projektverzeichnis ein versteckter Ordner `.git/` angelegt. Dieser Ordner enthält alle Versionsinformationen und stellt somit das lokale Repository dar.

Lokales Repository

Das lokale Repository bildet die Grundlage der Versionsverwaltung. Alle Änderungen an Projektdateien finden zunächst lokal statt. Um sie zu versionieren, werden diese in zwei Schritten übernommen:

1. **Staging Area**: Mit `git add .` werden Änderungen in die Staging Area übernommen.
2. **Commit**: Mit `git commit -m "Nachricht"` werden die Änderungen dauerhaft im lokalen Repository gespeichert.

Nachdem ein Projekt lokal eingerichtet wurde, kann es über IntelliJ mit einem Remote Repository verbunden werden. Dazu wird im Terminal der Remote-Link hinzugefügt: `git remote add origin https://github.com/Benutzername/Repositoryname.git` Dadurch wird spezifiziert, mit welchem entfernten Repository das lokale Projekt verknüpft ist.

Remote Repository

Ein Remote Repository liegt auf einem Server (z. B. GitHub oder GitLab) und ermöglicht die Zusammenarbeit im Team. Typischerweise wird auf der Plattform zunächst ein leeres Repository erstellt. Anschließend kann man das Remote Repository klonen `git clone <URL>`. Durch das Klonen entsteht automatisch ein vollständiges lokales Repository mit der Remote-Verknüpfung. Die Verbindung erfolgt in der Regel über eine HTTPS- oder SSH-URL, die im Git-Konfigurationsfile gespeichert wird.

Sobald die Verbindung steht, können Änderungen einfach synchronisiert werden:

1. **Push**: Überträgt lokale Commits auf das Remote Repository `git push`
2. **Pull**: Holt aktuelle Änderungen anderer Teammitglieder `git pull`

Vorteile der Git-Integration in IntelliJ sind:

- visuelle Darstellung des Versionsverlaufs und der Branch-Struktur
- Einfache Konfliktauflösung über integrierte Merge-Tools
- Automatische Erkennung von Änderungen und Statusanzeigen direkt im Editor
- Nahtlose Verbindung zu Git-Plattformen wie GitHub und GitLab

> Kursmaterial „Distributed Version Control Systems Vertiefung“, Kapitel 6.3 - 6.5.3

Nützliche Git-Tools und Plattformen

Git Plattformen

- GitHub

> GitHub ist eine proprietäre und öffentliche Softwareentwicklungsplattform auf Grundlage der Versionsverwaltungs-Software Git. GitHub war im Jahr 2011 bei Open-Source-Software der populärste Dienst seiner Art, gemessen an der Anzahl der Codebeiträge („Commits“). Der Dienst hat über 83 Millionen registrierte Nutzer und verwaltet über 200 Millionen Repositories (Stand: August 2022). Neben vielen sehr kleinen oder oft nur vom Besitzer genutzten Projekten gibt es mehrere bekannte größere Open-Source-Projekte, die bei der Versionsverwaltung ihres Quelltextes GitHub verwenden. Das Unternehmen GitHub, Inc. hat seinen Sitz in San Francisco in den USA und gehört zu Microsoft.

> Quelle: https://en.wikipedia.org/wiki/GitHub

- GitLab

> GitLab ist eine Softwareentwicklungsplattform auf Git-Basis. GitLab bietet ein Issue-Tracking-System mit Kanban-Board, ein System für Continuous Integration und Continuous Delivery (CI/CD), ein Wiki, eine Container-Registry, einen Sicherheitsscanner für Container und Sourcecode sowie Multi-Cluster-Verwaltung und -Überwachung. GitLab ist in Produkte für Entwickler, wie zum Beispiel AWS oder Google Cloud, integrierbar und über eine API fernsteuerbar. GitLab ist in den Programmiersprachen Ruby und Go geschrieben.
>
> Die GitLab Community Edition (CE) wird als Open-Source-Software unter der MIT-Lizenz entwickelt. Seit August 2013 bietet die GitLab Inc. mit der Enterprise Edition (EE) eine speziell für Unternehmen entwickelte Version an. Man kann GitLab auf eigener Hardware betreiben oder seit 2012 auf GitLab.com als GitLab Enterprise Edition – als Software as a Service (SaaS). Neben kostenlosem Hosting privater und öffentlicher Repositorys wird auch kostenpflichtiger Support angeboten.
>
> Quelle: https://de.wikipedia.org/wiki/GitLab

Viele Hochschulen bieten ihren Studierenden und Mitarbeitenden eigens gehostete GitLab-Instanzen an, z. B.:

  - GitLab an der THL
  - GitLab an der BHT

Git Clients

- GitHub Desktop

> Experience Git without the struggle
>
> Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.
>
> Quelle: https://desktop.github.com

- Sourcetree

> A free Git client for Windows and Mac
>
> Sourcetree simplifies how you interact with your Git repositories so you can focus on coding. Visualize and manage your repositories through Sourcetree's simple Git GUI.
>
> Quelle: https://www.sourcetreeapp.com

- GitKraken Desktop

> Simplifying Git for any OS
>
> Countless developers and teams worldwide use GitKraken Desktop for its intuitive GUI, powerful terminal, and cross-platform support for Windows, Mac, and Linux.

> Quelle: https://www.gitkraken.com/git-client

Git Lernressourcen

- Learn Git Branching - Interaktives Git-Lernspiel

> Interested in learning Git? Well you've come to the right place! "Learn Git Branching" is the most visual and interactive way to learn Git on the web; you'll be challenged with exciting levels, given step-by-step demonstrations of powerful features, and maybe even have a bit of fun along the way.
>
> Quelle: https://learngitbranching.js.org/

- Git Explained in 100 Seconds von Fireship - Kurzvideo über Git

> Learn the basics of Git in 100 seconds.
>
> Quelle: https://www.youtube.com/watch?v=hwP7WQkmECE&t=9s

- Git and GitHub for Beginners - Crash Course von freeCodeCamp - 1-stündiger Crash-Kurs über Git und GitHub

> Learn about Git and GitHub in this tutorial. These are important tools for all developers to understand. Git and GitHub make it easier to manage different software versions and make it easier for multiple people to work on the same software project.
>
> Quelle: https://www.youtube.com/watch?v=RGOj5yH7evk&t=70s

- Awesome Git - Eine kuratierte Liste von Git-Ressourcen

> A curated list of amazingly awesome Git tools, resources and shiny things.
>
> Quelle: https://github.com/dictcp/awesome-git