Git-Schulung – Einsteiger

Präsenz · 4 Stunden

Version: 1.0
Zielgruppe: Einsteiger
Format: Präsenz / Selbststudium

Inhaltsverzeichnis

Ziel der Schulung

Nach dieser Schulung können die Teilnehmer: - Git im Alltag sicher nutzen - Änderungen sauber versionieren - Branches verstehen und anwenden - Mit GitHub / GitLab arbeiten - Typische Fehler selbst beheben

Wichtig: Fokus auf Praxis, kein unnötiger Theorieballast.

Voraussetzungen

- Laptop (Windows / macOS / Linux)
- Internetzugang
- Texteditor (VS Code empfohlen, aber nicht Pflicht)
- Keine Git-Vorkenntnisse nötig

1. Einstieg – Was ist Git? (ca. 20 Minuten)

Was ist das Problem ohne Git?

Viele Projekte starten ohne Versionsverwaltung. Typische Situationen: - Mehrere Kopien eines Projekts: `projekt_final`, `projekt_final2`, `projekt_final_neu` - Niemand weiß, welche Version aktuell ist - Änderungen lassen sich nicht sauber zurückverfolgen - Fehler lassen sich nicht einfach rückgängig machen

Was Git löst

Git speichert den **Verlauf** eines Projekts: - Jede Änderung ist dokumentiert - Man kann jederzeit zu einem früheren Stand zurück - Mehrere Personen können parallel arbeiten

Was Git ist

- Ein **lokales Versionsverwaltungssystem**
- Läuft auf deinem Rechner
- Funktioniert auch **ohne Internet**

Was Git nicht ist

- Kein Backup-System
- Kein GitHub
- Kein Projektmanagement-Tool

Merksatz

> Git ist eine Zeitmaschine für Dateien.

Wissenswertes: Was bedeutet „Git“?

Git ist kein Akronym, sondern einfach der Name eines Werkzeugs zur Versionsverwaltung.
Der Name stammt von **Linus Torvalds**, dem Erfinder von Git.
Er sagte dazu sinngemäß:
„Ich bin ein egoistischer Bastard und nenne alle meine Projekte nach mir.“
Im britischen Englisch bedeutet **„git“** umgangssprachlich:

- Nervensäge
- Idiot
- Trottel

→ ironisch gemeint, typisch Linus.

2. Git installieren & erstes Repository (30 Minuten)

Installation

Git muss einmalig installiert werden.

Windows - Download von git-scm.com - Standardoptionen sind ausreichend

macOS - Installation über Xcode Command Line Tools oder Homebrew

Linux - Installation über den Paketmanager

Prüfen der Installation

```
git --version
```

Wenn eine Versionsnummer erscheint, ist Git korrekt installiert.

Erstes Repository erstellen

Ein Repository ist ein Ordner, der von Git überwacht wird.

```
mkdir git-demo
cd git-demo
git init
```

Nach `git init` erstellt Git einen versteckten Ordner `.git`. Dort speichert Git die komplette Historie.

Wichtige Begriffe

- **Repository**: Ein Projekt unter Git-Kontrolle
- **Working Directory**: Dein aktueller Arbeitsordner
- **Commit**: Ein gespeicherter Zustand

3. Die wichtigsten Git-Befehle (60 Minuten)

Grundprinzip

Git arbeitet in drei Stufen: 1. Dateien ändern 2. Änderungen vormerken 3. Änderungen speichern

`git status`

Zeigt jederzeit: - Welche Dateien geändert wurden - Welche Dateien für den Commit vorgemerkt sind

```
git status
```

`git add`

Markiert Dateien für den nächsten Commit.

```
git add main.c
```

Oder alle Dateien:

```
git add .
```

`git commit`

Speichert den aktuellen Stand dauerhaft.

```
git commit -m "Initiale Version"
```

`git log`

Zeigt die Commit-Historie.

```
git log --oneline
```

Typischer Ablauf

```
git status
git add .
git commit -m "Beschreibung der Änderung"
```

4. Branches verstehen (45 Minuten)

Was ist ein Branch?

Ein Branch ist ein **alternativer Entwicklungszweig**. - Er zeigt auf einen bestimmten Commit - Änderungen im Branch beeinflussen `main` nicht

Warum Branches sinnvoll sind

- Neue Funktionen testen
- Fehler beheben
- Experimente ohne Risiko

Branch erstellen und wechseln

```
git switch -c feature-test
```

Änderungen im Branch

- Dateien ändern
- Commit erstellen

Zurück nach `main` und mergen

```
git switch main
git merge feature-test
```

Merge-Konflikte

Konflikte entstehen, wenn: - Die gleiche Zeile in zwei Branches geändert wurde

Git stoppt dann und bittet um eine Entscheidung.

5. Git vs. GitHub vs. GitLab

Git

Git ist ein **lokales Versionsverwaltungssystem**. - Läuft auf deinem Rechner - Funktioniert ohne Internet - Speichert die komplette Projekt-Historie

Git kann: - Commits - Branches - Merges - Tags

Git kann **nicht**: - Zusammenarbeit im Web - Benutzerverwaltung - Issue-Tracking

GitHub

GitHub ist eine **Online-Plattform** für Git-Repositories.

Eigenschaften: - Web-Oberfläche - Zusammenarbeit im Team - Pull Requests - Issues - CI/CD (GitHub Actions)

Typische Nutzung: - Open-Source-Projekte - Private Projekte - Kleine bis mittlere Teams

GitLab

GitLab ist ebenfalls eine Git-Plattform, aber stärker auf **Teams und Firmen** ausgelegt.

Eigenschaften: - Cloud **oder** selbst hostbar - Integriertes CI/CD - Benutzer- und Rechteverwaltung

Typische Nutzung: - Firmenprojekte - Interne Repositories - Embedded- und Industrieprojekte

Vergleich

Thema	Git	GitHub	GitLab
Läuft lokal	☑	✗	✗
Internet nötig	✗	☑	☑
Versionsverwaltung	☑	☑	☑
Web-Oberfläche	✗	☑	☑
Teamarbeit	✗	☑	☑
Selbst hosten	✗	✗	☑
CI/CD	✗	☑	☑

Merksatz

> Git ist das Werkzeug – GitHub und GitLab sind die Plattformen.

6. Arbeiten mit Remote (GitHub / GitLab) (ca. 45 Minuten)

Was ist ein Remote?

Ein Remote ist ein Repository auf einem Server. - GitHub - GitLab - Firmeninterner Server

Wichtige Begriffe

- **clone**: Projekt herunterladen
- **pull**: Änderungen holen
- **push**: Änderungen senden

Repository klonen

```
git clone <repo-url>
```

Änderungen holen

```
git pull
```

Änderungen senden

```
git push
```

Typische Fehler

- Push abgelehnt → vorher `git pull`
- Falscher Branch → Branch prüfen

7. Fehler beheben & Sicherheit (ca. 35 Minuten)

Änderungen an Dateien verwerfen

```
git restore datei.txt
```

Letzten Commit korrigieren

```
git reset --soft HEAD~1
```

Änderungen rückgängig machen (sicher)

```
git revert <commit-id>
```

Wichtige Regel

Was gepusht wurde, sollte nicht mit reset geändert werden.

8. Tags & Versionen (v1.0, v1.1)

Wozu Tags?

Tags markieren **feste Versionen** im Projektverlauf.

Typische Einsatzfälle: - Firmware-Release (v1.0, v1.1) - Software-Versionen - reproduzierbare Builds

Ein Tag zeigt immer auf **einen bestimmten Commit**.

Tag erstellen

```
git tag v1.0
```

Tag mit Beschreibung (empfohlen)

```
git tag -a v1.1 -m "Bugfix Release"
```

Tags anzeigen

```
git tag
```

Tags pushen

Standardmäßig werden Tags **nicht automatisch** gepusht.

```
git push origin v1.1
```

Oder alle Tags:

```
git push origin --tags
```

Embedded-Hinweis

- Tag = exakt reproduzierbarer Firmware-Stand
- Sehr wichtig für Support & Fehlersuche

9. Best Practices & Abschluss

Best Practices

- Kleine, saubere Commits
- Sinnvolle Commit-Messages
- `.gitignore` für:
  - Build-Ordner (Embedded)
  - `node_modules` (Web)

Abschluss

- Fragen klären
- Nächste Schritte aufzeigen

Anhang – Spickzettel

```
git status      – aktueller Zustand
git add         – Änderungen vormerken
git commit      – Änderungen speichern
git log         – Historie anzeigen
git switch      – Branch wechseln
git pull        – Änderungen holen
git push        – Änderungen senden
git tag         – Version markieren
```

==========

PDF-HINWEISE & STRUKTUR

==========

Trainer-Version vs. Teilnehmer-Version

Teilnehmer-Version

Diese Version ist für **Selbststudium und Nachschlagen** gedacht.

Merkmale: - Erklärtexte vollständig - Beispiele und Merksätze - Übungen ohne Lösung

Trainer-Version

Diese Version enthält **zusätzliche Hinweise für die Durchführung**.

Zusätzlich enthalten: - Trainer-Notizen - typische Fragen - empfohlene Reihenfolge - Zeitmanagement-Hinweise

Empfehlung: - Teilnehmer bekommen die Teilnehmer-Version - Trainer arbeitet mit der Trainer-Version

Übungsaufgaben

Kapitel 1 – Git verstehen

Trainer-Notizen

- Teilnehmern klar machen: Git schützt vor Datenverlust
- Angst nehmen: Git kann fast nichts „kaputt machen"

Übung 1: - Erkläre in eigenen Worten den Unterschied zwischen Git und GitHub - Warum funktioniert Git auch ohne Internet?

Kapitel 2 – Repository

Trainer-Notizen

- `.git` nicht anfassen oder löschen
- Betonung: Git verändert Dateien nicht automatisch

Übung 2: 1. Lege ein neues Verzeichnis an 2. Initialisiere ein Git-Repository 3. Prüfe den Status

---

Kapitel 3 – Commits

Trainer-Notizen

- Immer wieder `git status` zeigen
- Commit-Messages erklären wie ein Änderungsprotokoll

Übung 3: 1. Lege eine Datei an (`main.c` oder `index.html`) 2. Erstelle drei Commits mit sinnvollen Nachrichten 3. Zeige die Historie an

---

Kapitel 4 – Branches

Trainer-Notizen

- Konflikte bewusst erzeugen
- Zeigen, dass Konflikte lösbar sind

Übung 4: 1. Erstelle einen Feature-Branch 2. Ändere eine Datei 3. Merge den Branch zurück nach `main`

Kapitel 5 – GitHub / GitLab

Trainer-Notizen

- Diese Abgrenzung nimmt viel Verwirrung
- Erst Git erklären, **dann** Plattform

Übung 5: 1. Klone ein Repository 2. Ändere eine Datei 3. Push die Änderung

---

Kapitel 6 – Fehler beheben

Trainer-Notizen

- HTTPS für Einsteiger empfehlen
- Remote als “gemeinsamen Treffpunkt” erklären

Übung 6: 1. Ändere eine Datei 2. Verwerfe die Änderung mit `git restore`

Kapitel 7 – Tags

Trainer-Notizen

- Unterschied reset vs. revert klar machen
- Sicherheit betonen

Trainer-Notizen

- Tags sind **read-only Markierungen**
- Nicht für tägliche Entwicklung nutzen

Übung 7: 1. Erstelle einen Tag `v1.0` 2. Erstelle einen annotierten Tag `v1.1` 3. Zeige alle Tags an

Release-Workflow – Tag → ZIP → Weitergabe

Ziel

Einen **klar definierten Stand** weitergeben (z. B. Firmware oder Web-Release).

Schritt 1 – Sauberen Stand prüfen

```
git status
```

Ergebnis sollte sein:

```
working tree clean
```

Schritt 2 – Tag setzen

```
git tag -a v1.0 -m "Release v1.0"
```

Schritt 3 – Tag pushen

```
git push origin v1.0
```

Schritt 4 – ZIP aus Tag erzeugen

```
git archive --format=zip v1.0 -o projekt-v1.0.zip
```

Ergebnis

- ZIP enthält **exakt** den getaggten Stand
- Keine Build-Artefakte
- Reproduzierbar

Embedded-Hinweis

- Ideal für Firmware-Weitergabe
- Support kann immer exakt diesen Stand auschecken