Versionierung

Die Versionsnummer setzt sich aus drei Zahlen zusammen.

Die erste Zahl ist der **Major** = inkompatible Änderung

Die zweite Zahl ist die **Minor** = neue Features, abwärtskompatibel

Die dritte Zahl ist der **Patch** = Bugfixes, abwärtskompatibel

Warum nutzen wir Versionierung?

- Um Änderungen zu sehen
- Fehler finden vereinfachen
- Nachvollziehbarkeit

GIT

1. **Initialisierung / Projektstart**
    a. Ein Projekt wird als Repository angelegt, dass alle Dateien und die gesamte Versionshistorie speichert.
    b. Man unterscheidet zwischen lokales Repo (auf dem Entwickler-PC) und Remote Repo (Github / Gitlab).
2. **Branching Konzept**
    a. Main: Stabile Hauptlinie, aus der produktive Software entsteht.
    b. Feature Branches: Für neue Funktionen und Änderungen
    c. Bugfixes/Hotfix Branches: Für schnelle Fehlerkorrektur
    d. Release Branches(optional): Vorbereitung auf neue Version
3. **Arbeiten im Projekt**
    a. Änderungen werden auf einem eigenen Branches entwickelt
    b. Jede Änderung wird durch einen Commit gespeichert, der aus zwei Teilen besteht: Inhalt (geänderten Dateien) / Metadaten (Autor/Datum/Commit-Message).
    c. So entsteht eine Historie, die jeden Entwicklungsschritt nachvollziehbar macht
4. **Zusammenarbeiten im Team**
    a. Entwickler synchronisieren ihr lokales Repo regelmäßig mit dem Remote Repo
    b. Wenn ein Branch fertig ist, wird er über einen Merge oder Rebase in den Hauptzweig integriert.
    c. Üblich ist ein Pull Request(PR) oder Merge Request(MR)
        i. Dort können anderen Entwickler änderungen prüfen (Code Review)
        ii. Automatisierte Tests laufen zur Qualitätssicherung.

5. **Konflikte**
    a. Wenn zwei Entwickler dieselben Stellen im code ändern, ensteht ein Konflikt
    b. Dieser muss manuell beseitigt werden.
6. **Release Management**
    a. Sobald ein Hautzweig stabil ist, wird ein Release erstellt
    b. Dies geschieht häufig durch Tagging. (1.2.0)
    c. Dies ermöglicht es, jederzeit auf diesen definierten Stand zurückzuspringen.
7. **Vorteile eines strukturierten Git-Workflows**
    a. Nachvollziehbarkeit = Wer hat Wann Was geändert
    b. Teamarbeit = Mehrere Entwickler können parallel arbeiten
    c. Qualitätssicherung = Code Reviews, Tests
    d. Sicherheit = Fehler lassen sich durch ein zurückspringen auf eine frühere Version beheben.

Zusammenfassung

Ein Git Projekt läuft theoretisch so ab:

Repos erstellen -> Branches nutzen -> Commits schreiben -> Branches zusammenführen -> Release taggen.