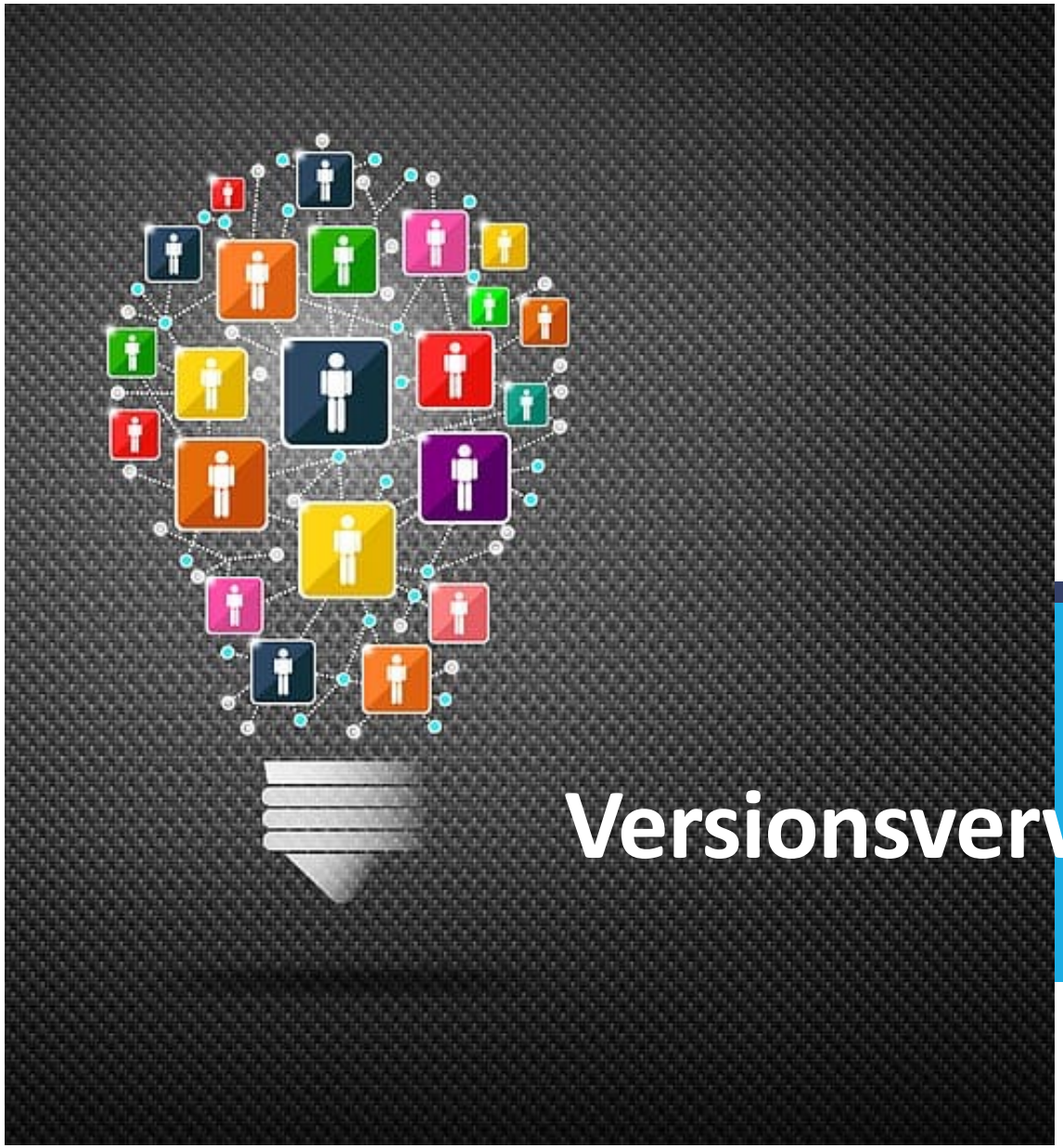


Versionsverwaltung

Inhalt

- Versionsverwaltungssystem
- Funktionen
- Workflows

Übersicht

Versionsverwaltungssystem

- Begriffsklärung
- Zentral versus Dezentral
- Eigenschaften verschiedener Versionsverwaltungssysteme

Versionsverwaltungssystem

- Software-Werkzeug, das Änderungen an Dateien dokumentiert und verwaltet
- wird verwendet, um den Entwicklungsstand von Quellcode nachzuvollziehen, frühere Versionen wiederherzustellen und Zusammenarbeit im Team zu ermöglichen

Zentral versus Dezentral

Zentrale Versionsverwaltung (CVCS)

speichert alle Projektversionen auf einem zentralen Server

Vorteile:

- Einfache Handhabung für kleinere Teams
- Klare Nachverfolgung von Änderungen (wer hat wann was geändert)
- Einfache Verwaltung von Zugriffsrechten

Nachteile:

- Abhängigkeit vom zentralen Server. Funktioniert nicht, wenn der Server offline ist
- Höhere Latenz bei der Arbeit mit großen Dateien oder über langsame Netzwerke
- Geringere Flexibilität und Autonomie für Entwickelnde

Beispiele: TFS, CVS, Subversion (SVN), Perforce

Dezentrale Versionsverwaltung (DVCS)

jedem Entwickelndem wird eine vollständige Kopie des Projekts zur Verfügung gestellt

Vorteile:

- Jeder Entwickelnde hat eine vollständige Kopie des Projekts, was Offline-Arbeit ermöglicht
- Höhere Flexibilität und Autonomie für Entwickler
- Schnellere lokale Operationen, da keine Netzwerkverbindung erforderlich ist
- Bessere Skalierbarkeit für größere Teams und Projekte

Nachteile:

- Komplexere Verwaltung, insbesondere für Anfänger
- Potenziell höhere Speichernutzung auf den einzelnen Entwickler-Rechnern

Beispiele: Git, Mercurial

Eigenschaften verschiedener Versionsverwaltungssysteme

Eigenschaft	SVN (Subversion)	CVS	TFS mit SourceSafe	Git
Typ	Zentral	Zentral	Zentral (TFS), lokal (SourceSafe)	Dezentral
Lizenz	Open Source	Open Source	Proprietär (Microsoft)	Open Source
Plattformunterstützung	Windows, Linux, macOS	Windows, Linux, macOS	Windows (fokussiert)	Windows, Linux, macOS
Offline-Arbeit möglich?	Eingeschränkt	Nein	Teilweise (mit SourceSafe)	Ja
Branching/Merging	Möglich, aber aufwendig	Kaum brauchbar	Eingeschränkt (v. a. in TFS)	Sehr flexibel & leistungsstark
Eignung für Teams	Gut für kleine/mittlere Teams	Veraltet – nicht mehr empfohlen	Gut in Microsoft-zentrierten Umgebungen	Exzellent, auch für große Teams
Beliebtheit (2025)	Mittel	Gering (historisch wichtig)	Gering außerhalb von MS-Umgebungen	Sehr hoch
Vorteile	Einfach, zentral, gute Kontrolle	Schnell, leichtgewichtig	Integration in MS-Toolchain	Schnell, flexibel, riesige Community
Nachteile	Wenig flexibel, kein Offline-Modus	Veraltet, schlechte Merge-Funktionen	Eingeschränkt auf Microsoft-Umgebung	Komplexe Einstiegshürde für Anfänger

Übersicht

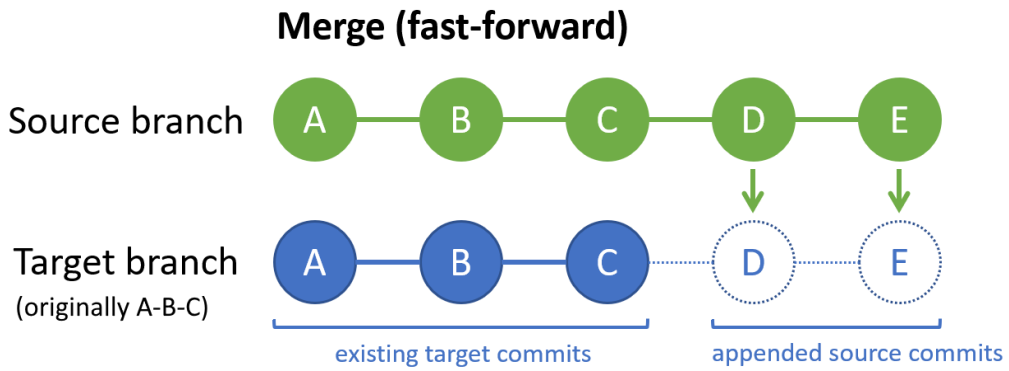
Funktionen

- Überblick
- Merge
- Reset
- Checkout
- Rebase
- Revert

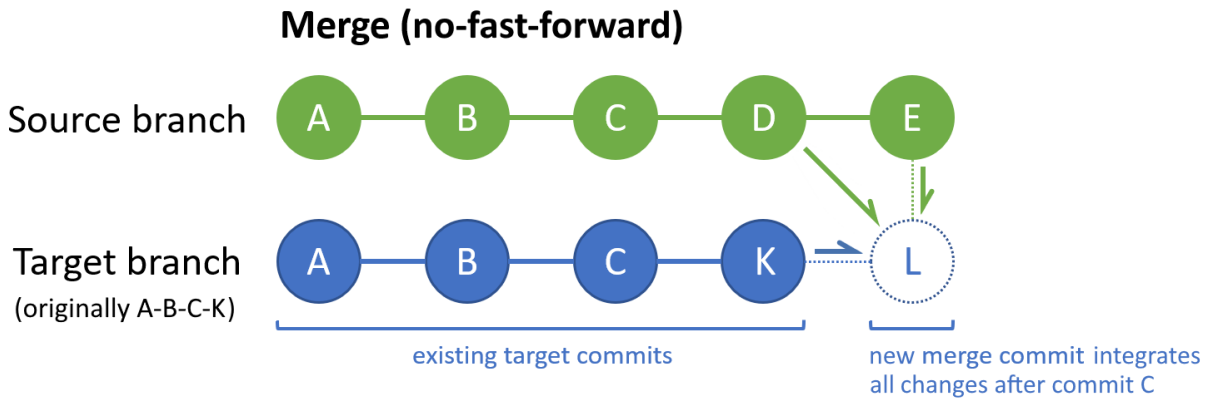
Funktionen

Funktion	Beschreibung	Typischer Anwendungsfall
Commit	Speichert eine Änderung im lokalen Repository mit einer Nachricht zur Beschreibung des Inhalts.	Zwischenschritt im Entwicklungsprozess festhalten
Revert	Macht einen oder mehrere frühere Commits durch einen neuen „Gegen-Commit“ rückgängig.	Fehlerhafte Änderung rückgängig machen, ohne die Historie zu zerstören
Branch	Erstellt einen neuen Entwicklungszweig vom aktuellen Stand.	Neue Features oder Bugfixes parallel zur Hauptentwicklung umsetzen
Merge	Führt zwei Branches zusammen; Änderungen beider werden kombiniert.	Feature-Branch in den Hauptbranch (main/master) integrieren
Cherry-Pick	Holt sich gezielt einzelne Commits aus einem anderen Branch.	Nur ausgewählte Änderungen übernehmen, ohne gesamten Branch zu mergen
Pull	Holt die neuesten Änderungen vom Remote-Repository und integriert sie lokal (Fetch + Merge).	Projektstand mit dem Server synchronisieren
Push	Sendet lokale Commits zum Remote-Repository.	Eigene Änderungen für andere im Team verfügbar machen
Rebase	„Umschreibt“ die Commit-Historie, indem Änderungen auf einen neuen Basis-Branch angewendet werden.	Aufräumen und linearisieren der Historie, z. B. vor dem Merge

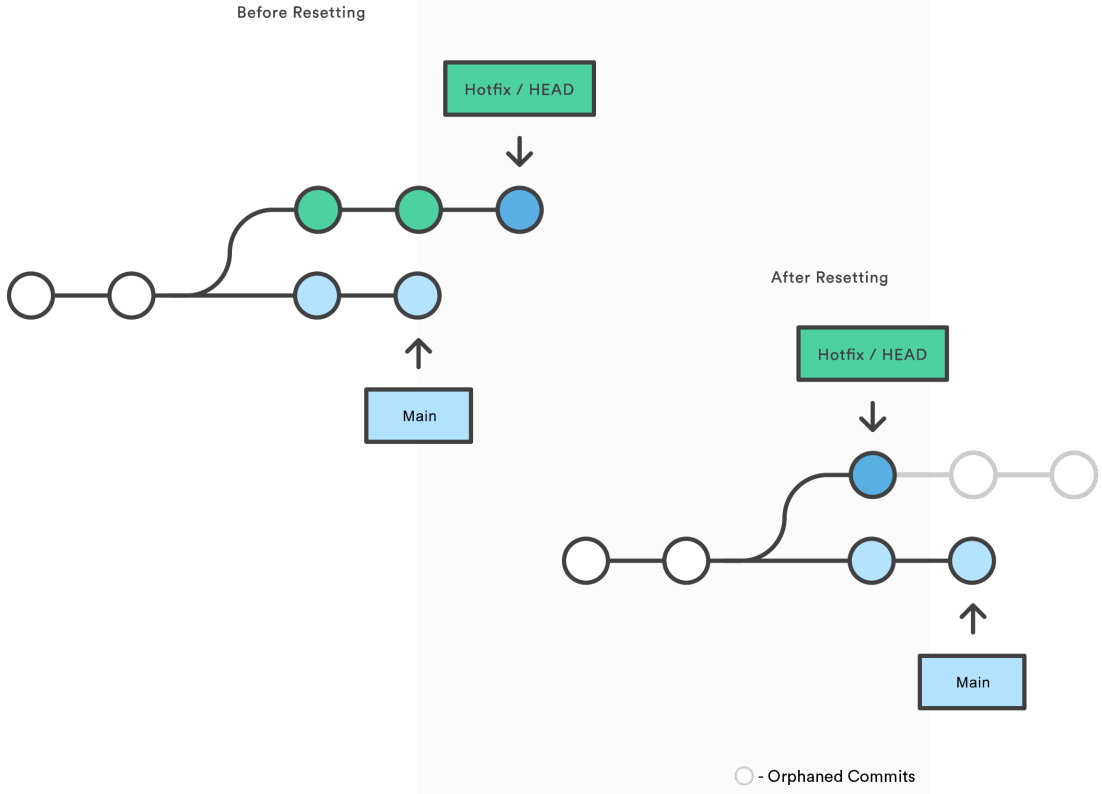
Merge
(fast-forward)



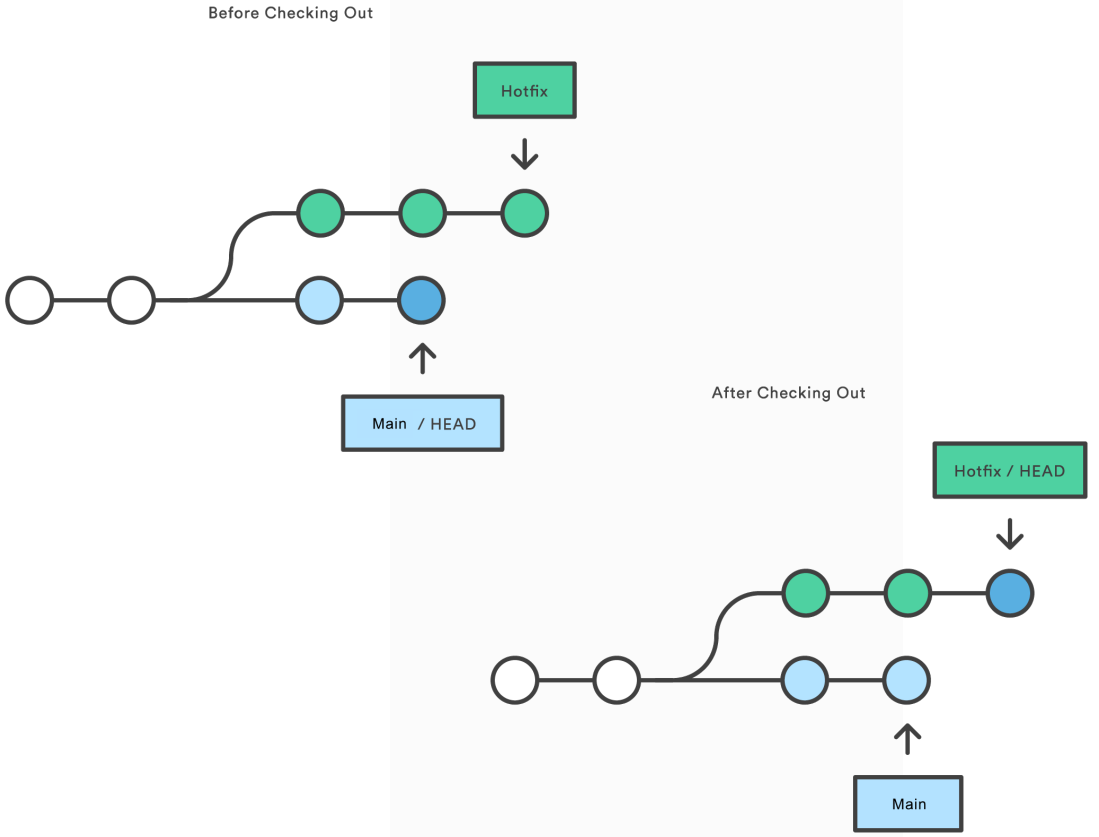
Merge
(no-fast-forward)



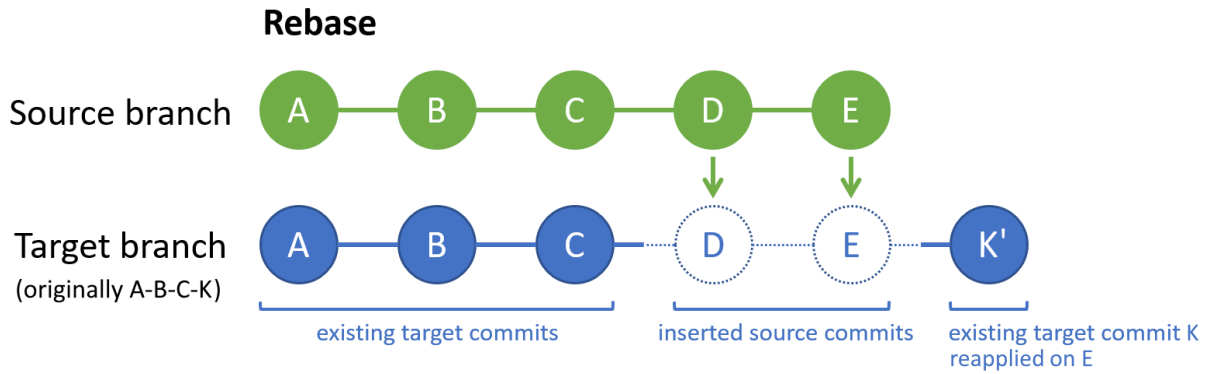
Reset


Before Resetting
Hotfix / HEAD
Main
After Resetting
Hotfix / HEAD
Main
- Orphaned Commits

Checkout

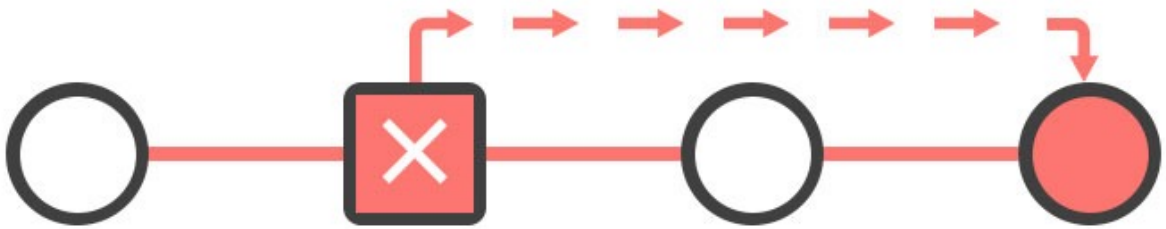

Before Checking Out
Hotfix
Main / HEAD
After Checking Out
Hotfix / HEAD
Main

Rebase


Rebase
Source branch
A
B
C
D
E
Target branch
(originally A-B-C-K)
A
B
C
D
E
K'
existing target commits
inserted source commits
existing target commit K
reapplied on E

Revert

Reverting



Cherry Pick


Main Branch
Feature Branch
Commit 2c
Commit 2b
Cherry Pick
Commit 2a

Übersicht

Workflows

- Überblick
- Centralized Workflow
- Feature-Branch-Workflow
- Git Flow
- Forking Workflow
- Trunk-Based Development
- Pull Request

Workflows

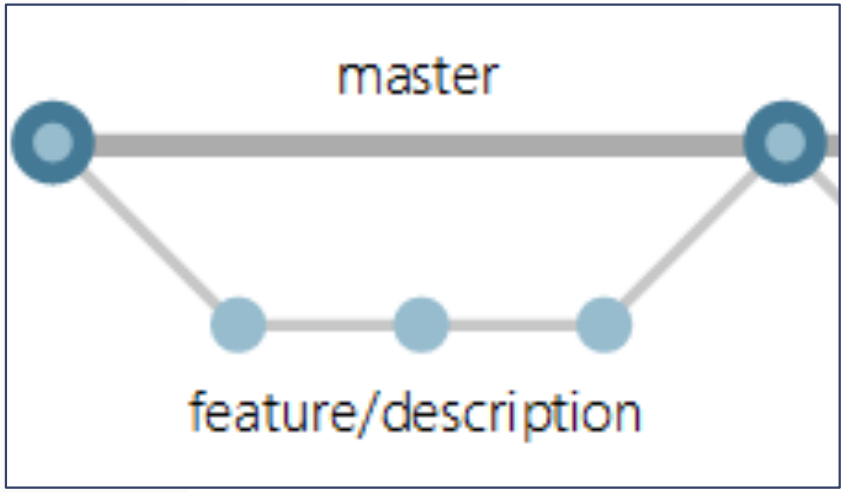
Workflow	Teamgröße	Komplexität	CI/CD-freundlich	Hauptvorteil
Centralized Workflow	1–5 Personen	Niedrig	Bedingt	Einfach und direkt
Feature-Branch	2–15 Personen	Mittel	Ja	Getrennte Entwicklung
Git Flow	10+ Personen	Hoch	Ja	Strukturierter Release-Prozess
Forking Workflow	5–500+ (extern)	Mittel	Ja	Sichere Zusammenarbeit
Trunk-Based Development	3–50 Personen	Hoch	Ja	Hohe Geschwindigkeit & CI/CD

Centralized Workflow (klassischer SVN-ähnlicher Stil)

- **Prinzip**: Alle Entwickelnde arbeiten direkt im Haupt-Branch (z. B. main)
- **Ablauf**:
  pull → bearbeiten → commit → push
- **Vorteile**:
  - Einfach zu verstehen
  - Gut für kleine Teams
- **Nachteile**:
  - Keine parallelen Entwicklungen
  - Konfliktanfällig bei mehreren Entwickelnden

Feature-Branch-Workflow

- **Prinzip**: Jede neue Funktion wird in einem eigenen Branch entwickelt
- **Ablauf**:
  main → feature/login → Entwicklung → Merge in main (per Pull Request)
- **Vorteile**:
  - Klare Trennung von Features
  - Verbesserte Code-Reviews durch Pull Requests
- **Nachteile**:
  - Merge-Konflikte möglich
  - Leicht komplexer für Anfänger

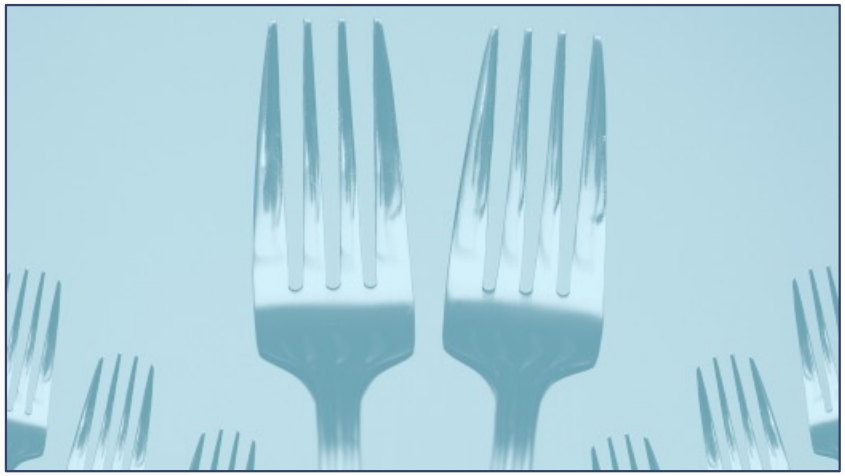


Git Flow

- **Prinzip**: Strukturierter Workflow mit langfristigen Branches (main, develop) und temporären Branches (feature, release, hotfix)
- **Ablauf**:
  - main: Stabile Releases
  - develop: Integration neuer Features
  - feature/*: Neue Features
  - release/*: Vorbereitung auf Release
  - hotfix/*: Notfallkorrekturen
- **Vorteile**:
  - Klare Prozesse für Teams
  - Skalierbar für große Projekte
- **Nachteile**:
  - Komplexer Aufbau
  - Manchmal Overhead für kleinere Projekte

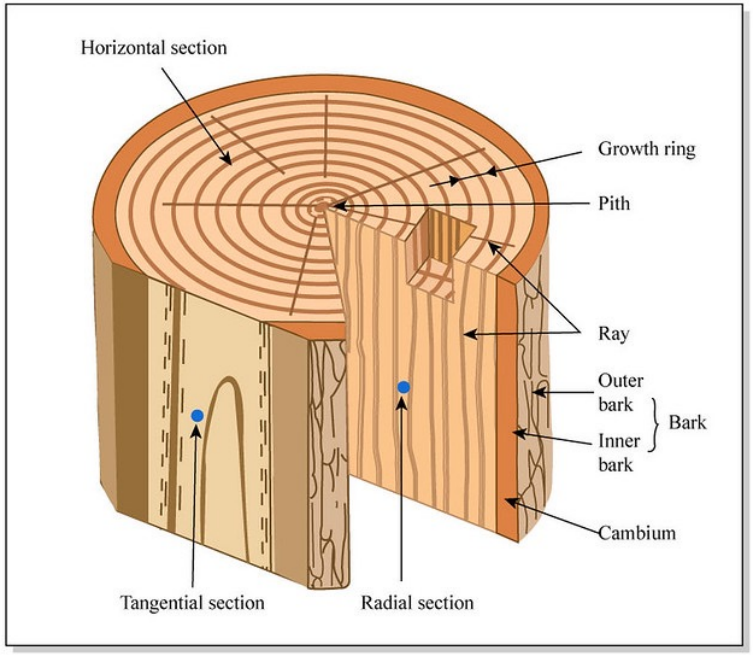
Forking Workflow

- **Prinzip**: Entwickelnde forken das zentrale Repository und senden Pull Requests von ihrem eigenen Repository
- **Vorteile**:
  - Hohe Sicherheit (niemand schreibt direkt ins Haupt-Repo)
  - Gut für externe Mitwirkende
- **Nachteile**:
  - Weniger direkt für interne Teams
  - Review- und Merge-Prozess notwendig



Trunk-Based Development

- **Prinzip**: Alle arbeiten in einem einzigen Branch (z. B. main), machen jedoch **sehr häufige, kleine Commits, ergänzt durch:**
  - **Feature-Toggles** (um unfertige Features zu verstecken)
  - **CI/CD** für automatisierte Tests und Deployments
- **Vorteile**:
  - Schnelles Feedback, hohe Integrationsfrequenz
  - Geeignet für DevOps-Teams
- **Nachteile**:
  - Erfordert Disziplin & gute Testabdeckung
  - Nicht für jedes Team geeignet



Begriffsklärung: Pull Request

- **formelle Anfrage**, Änderungen in einen Ziel-Branch zu integrieren
- andere Teammitglieder können den Code überprüfen, diskutieren und ggf. Verbesserungen vorschlagen – bevor die Änderungen „gemerged“ werden

- **Typischer Ablauf**