# Version Control

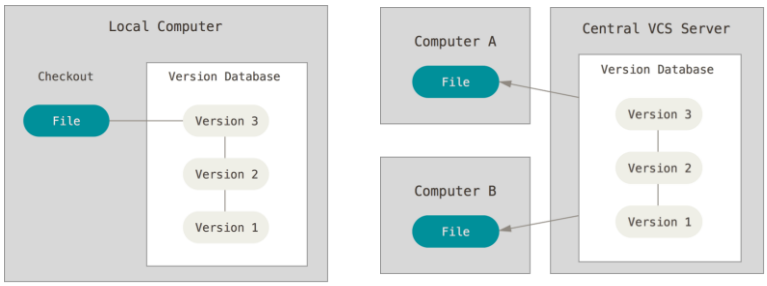
## Definition

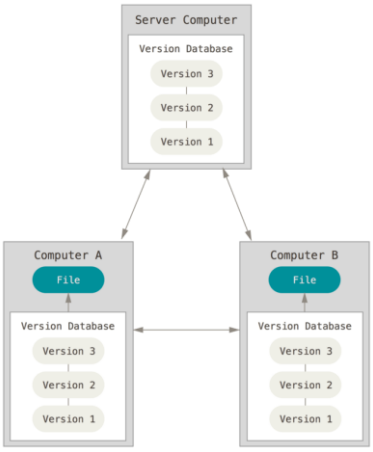
Unter Versionskontrolle bzw. Versionsverwaltung versteht man eine Software, welche dazu dient, eine oder mehrere Dateien oder Code zu verwalten, um deren einzelne Versionen zurückverfolgen zu können.

| Ziel | Aufgabe |
|---|---|
| Eindeutigkeit bezüglich der aktuellen Version | Koordinierung & Ermöglichung gemeinsamer Zugriffe & Entwicklungen durch verschiedener Entwickler |
| Nachvollziehbarkeit der Versionen | Protokollierung aller Änderungen |
| Datensicherung | Wiederherstellung von älteren Ständen & Archivierung einzelner Stände |

## Arten der Versionskontrolle

1. Lokale Versionskontrolle
- Lokale Versionierung auf einem Rechner, meist nur für eine Datei
2. Zentrale Versionskontrolle
- Basierend auf einem Client-Server-System -> Versionen werden bei einem Commit in einem gemeinsamen Repository auf einem Server gespeichert
- Keine lokale Versionshistorie möglich & Änderungen nur über ein Netzwerk machbar
3. Verteilte / dezentrale Versionskontrolle
- Jeder Entwickler hat sein eigenes Repository & kann es mit den Repositories der anderen abgleichen
- Durch die Möglichkeit der gleichzeitigen Arbeit an einer Datei kommt es zur Entstehung verschiedener Versionshistorien, die unteranderem parallel verlaufen
- Ermöglicht es offline Versionen zu erstellen & Änderungen vorzunehmen
- Das Teilen der Repositories muss nicht über einen zentralen Server ablaufen

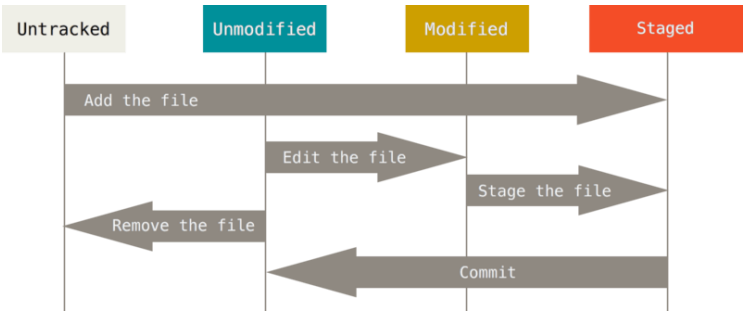



## Git

- Open-Source Software zur verteilten Versionsverwaltung
- 2005 von Linus Trovalds entwickelt & heute die meist verbreitetste Software seiner Art

## Lifecycle einer Datei in Git




Untracked

- Standard beim Anlegen einer neuen Datei

Staged

- Markieren der Datei, damit diese mit dem nächsten Commit versioniert wird

Unmodified

- Die Version der Datei ist getrackt (in einem Commit vorhanden), wurde aber seitdem nicht verändert

Modified

- Die Version der Datei ist getrackt, jedoch wurde sie lokal verändert

## Terminologie

Repository

- Verzeichnis, in dem digitale Objekte (Code, Dokumente, usw.) gespeichert & verwaltet werden. Es dient als digitales Archiv.

Commit

- Synonym für eine Version eines Projekts. Der Begriff hat seinen Ursprung im gleichnamigen Befehl „commit“, welcher genutzt wird, um nach Änderungen eine neue Version in das Repository zu laden.

Branch

- Ein Verweis auf einen Commit des Projekts, wobei dieser auch Nachfolger habe kann. Somit stellt ein Branch eine Entwicklungslinie dar. Der neuste Commit eines Branches ist durch den „HEAD“ gekennzeichnet.

Merge

- Ein Befehl, der das Verschmelzen mehrerer parallel verlaufender Commits durchführt. Dies tritt nur bei der verteilten Versionskontrolle auf.

Push

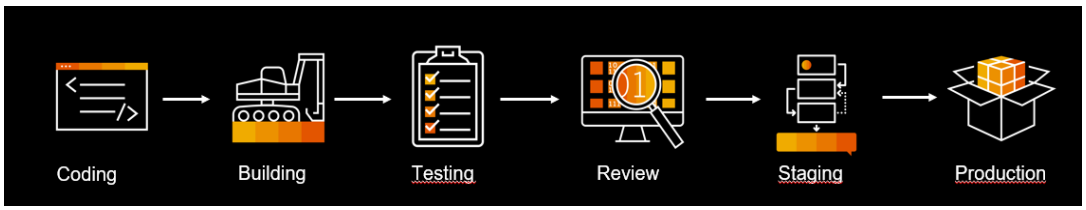
- Ein Git-Befehl, der verwendet wird, um lokale Commits in ein Remote-Repository zu übertragen.

Pull

- Ein Git-Befehl, der umgekehrt zu (git) push, funktioniert. Es werden Also Inhalte aus einem Remote-Repository heruntergeladen & damit das lokale Repository aktualisiert.

# CI/CD

## Continuous Integration und Continuous Delivery/Deployment



Beispiel einer CI/CD-Pipeline

## Wozu CI/CD?

- regelmäßige Bereitstellung von Apps
- Automatisierung aller Phasen in der Anwendungsentwicklung
- löst Probleme, die bei Integration von neuem Code auftreten
- führt zu Transparenz im App-Lifecycle

CI/CD nutzt das Konzept des Pipelining. Dabei werden Befehle in kleine schnell lösbare Teilaufgaben zerlegt, die auch parallel abgearbeitet werden. Dadurch durchlaufen die Codes den Lifecycle schneller und die Entwickler bekommen schneller Feedback.

## Continous Integration Pipeline:

- Fokus auf Integration neuen Codes
- jeder hat Zugriff auf Build und Testrepository
- mindestens tägliche Check-In des Codes
- Build ist schnell
- Validierung des Codes mit Unit Tests und Integration Tests
- Build-Artefakte werden nach erfolgreichem Build in Testumgebung deployt

Wenn Tests oder ein Build fehlschlägt, gilt es als Broken. Darüber werden die Entwickler sofort informiert, sobald die Tests durchlaufen sind. Dann gilt es die Fehler zu beheben, damit das Feature in die Testumgebung kann.

## Continuous Delivery Pipeline:

- Fokus auf Auslieferungsfähigkeit
- automatisiertes Feedback über Auslieferungszustand bei Änderungen
- Deployments beliebiger Version per Knopfdruck möglich
- Automatisierung aller Bereiche des Auslieferungsprozesses

## Continuous Deployment:

Unterschied zu Continuous Delivery:

- jede Änderung direkt in Produktion
    - Zeitfenster für verlorene Gelegenheit klein

Hier besteht nicht die Entscheidung dafür, ob die Änderung in die Produktion gehen soll oder nicht. Stattdessen ist der Deploy-Mechanismus ein Teil der automatisierten Pipline. Hierbei kann man jedoch seine Features auch toggeln, wodurch unfertige Teile im Produktivcode sein können, deren Funktionalität jedoch

nicht genutzt werden kann. Dadurch kann man Feedback über das Deployment der neuen Teile erhalten. Auch ist es möglich die neun Features zu Kleingruppen zu deployen, um Feedback von Nutzern zu erhalten.

Quellen:


1. https://www.scrum.de/untershiede-zwischen-continuous-integration-continuous-delivery-und-continuous-deployment/ [aufgerufen am 22.04.2021 12:00]
2. https://www.redhat.com/de/topics/devops/what-is-ci-cd [aufgerufen am 20.04.2021 13:00]
3. https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository [aufgerufen am 18.04.2021 14:30]
4. https://www.atlassian.com/de/git/tutorials/what-is-version-control [aufgerufen am 18.04.2021 14:13]
5. https://git-scm.com/docs/gitglossary [aufgerufen am 18.04.2021 19:10]
6. https://git-scm.com/book/de/v2/Git-Branching-Einfaches-Branching-und-Merging [aufgerufen am 18.04.2021 19:40]