

Digital Skills für Ingenieur*innen

11. Vorlesungstag

„Versionskontrollsystem Git“

Voraussetzungen zum Starten der Software

- Es ist wichtig, dass Sie die **Mindestvoraussetzungen** für ein Projekt beschreiben.
- Welche **Bibliotheken** zum Starten der Software sind notwendig?
- Welches **Betriebssystem** ist notwendig?
- Müssen **Daten für den Start der Software** bereitgestellt werden?
 - Falls ja, **in welchem Format müssen die Eingabedaten** bereitgestellt werden?

Versionskontrollsystem – Git

Was ist Versionverwaltung?

- Versionsverwaltung ist ein System, welches **die Änderungen an einer oder einer Reihe von Dateien über die Zeit hinweg protokolliert**, sodass man später auf eine **bestimmte Version zurückgreifen** kann.
- Es gibt grundsätzlich drei Arten von Versionsverwaltung
 - × **Lokale Versionsverwaltung**, z.B. kopieren von Quellcode in ein separates Verzeichnis
 - × **Zentrale Versionsverwaltung**, z.B. Subversion
 - × **Dezentrale Versionsverwaltung**, z.B. Git, Mercurial, Bazaar oder Darcs

Motivation für ein Versionskontrollsystem

- Typischer Software Entwicklungsprozess:
Erschaffen (Create) → Bearbeiten (Edit) → Speichern (Save) → Wiederholen (Repeat)
- Bei der Bearbeitung größerer Projekte möchte man oft folgende Dinge tun/wissen:
 - × **Verfolgen** der Projektgeschichte
 - × **Zurücknehmen** von Änderungen
 - × **Verteiltes, kollaboratives Arbeiten** am Projekt
 - × **Effiziente Verwaltung**
 - × **Unkomplizierte Veröffentlichung** des Quelltextes

Frühzeitige Versionierung mittels Tar

```
>ls versionen/  
Projekt.2022.01.15.tar  
Projekt.2022.01.16.tar  
Projekt.2022.08.12.tar  
Projekt.2022.05.21.tar  
Projekt.2022.05.17.tar  
...
```

Abbildung 1: Versionierung mittels Tar-Archiven
(Datum)

Frühzeitige Versionierung mittels Tar

```
>ls versionen/  
Projekt.2022.01.15.tar  
Projekt.2022.01.16.tar  
Projekt.2022.08.12.tar  
Projekt.2022.05.21.tar  
Projekt.2022.05.17.tar  
...
```

Abbildung 1: Versionierung mittels Tar-Archiven
(Datum)

Man bekommt das **Wann** und das **Was** mit, allerdings **nicht das Warum**.

Frühzeitige Versionierung mittels Tar

```
>ls versionen/  
Projekt.2022.01.15.Andreas.tar  
Projekt.2022.01.16.Markus.tar  
Projekt.2022.08.12.Christian.tar  
Projekt.2022.05.21.Klaus.tar  
Projekt.2022.05.17.Andreas.tar  
...
```

Abbildung 2: Versionierung mittels Tar-Archiven
(Datum + Name)

Frühzeitige Versionierung mittels Tar

```
>ls versionen/  
Projekt.2022.01.15.Andreas.tar  
Projekt.2022.01.16.Markus.tar  
Projekt.2022.08.12.Christian.tar  
Projekt.2022.05.21.Klaus.tar  
Projekt.2022.05.17.Andreas.tar  
...
```

Abbildung 2: Versionierung mittels Tar-Archiven
(Datum + Name)

Man bekommt nur das **Wer** heraus. Aber welche Version ist neuer?

Versionskontrollsystem – Git

Git Historie



Abbildung 3: Linus Torvalds
(2002)

Versionskontrollsystem – Git

Git Historie

- „Ich bin ein egoistischer Mistkerl, und ich benenne all meine Projekte nach mir. Zuerst ‚Linux‘, jetzt eben ‚Git‘.“



Abbildung 3: Linus Torvalds
(2002)

Git Historie

- „Ich bin ein egoistischer Mistkerl, und ich benenne all meine Projekte nach mir. Zuerst ‚Linux‘, jetzt eben ‚Git‘.“
- „Der Witz ‚Ich benenne alle meine Projekte nach mir, zuerst Linux, nun eben Git‘ war einfach zu gut, um ihn nicht zu machen. Aber es (der Befehl) ist auch kurz, einfach auszusprechen und auf einer Standardtastatur zu schreiben, dazu einigermaßen einzigartig und kein gewöhnliches Standardkommando – was ungewöhnlich ist.“



Abbildung 3: Linus Torvalds
(2002)

Versionskontrollsystem – Git

Git Historie

- „Ich bin ein egoistischer Mistkerl, und ich benenne all meine Projekte nach mir. Zuerst ‚Linux‘, jetzt eben ‚Git‘.“
- „Der Witz ‚Ich benenne alle meine Projekte nach mir, zuerst Linux, nun eben Git‘ war einfach zu gut, um ihn nicht zu machen. Aber es (der Befehl) ist auch kurz, einfach auszusprechen und auf einer Standardtastatur zu schreiben, dazu einigermaßen einzigartig und kein gewöhnliches Standardkommando – was ungewöhnlich ist.“
- Git (ugs.) [British] – Schwachkopf, widerlicher Penner



Abbildung 3: Linus Torvalds
(2002)

Git Historie

- Linus Torvalds begann mit der Entwicklung von Git im April 2005, da das von den Linux-Kernel-Entwickler*innen genutzte System BitKeeper nicht mehr kostenlos erhältlich und somit vielen Entwickler*innen der Zugang verwehrt blieb.
- Alle bereits bestehenden Systemen entsprachen nicht den Anforderungen von Torvalds (zu langsam, kompliziert, nicht für große Projekte geeignet oder nicht verteilt).
- Derzeitiger Maintainer von Git ist Junio Hamano.

Git Eigenschaften

- **Nicht-lineare Entwicklung**
Erstellen und Verschmelzen von Entwicklungszweigen; effiziente Implementierung der Zweige.
- **Kein zentraler Server**
Jeder Entwickler besitzt eine Arbeitskopie mit der gesamten Projektgeschichte. Alle Remotes und lokale Kopien sind technisch gleich.
- **Datentransfer zwischen Repositorys**
file://, git://, ssh://, http://, https://, ftp:// oder rsync://

Git Eigenschaften

- **Kryptografische Sicherheit der Projektgeschichte**
Der Hash eines einzelnen Commits basiert auf der vollständigen Geschichte, die zu diesem Commit geführt hat.
- **Speichersystem und Dateiversionierung**
Es erhält nicht jede Datei eine Revisionsnummer, wie bei CVS, sondern es werden Verweise ähnlich einem Dateisystem gespeichert. **Ändert sich eine Datei nicht, ist kein erneutes Speichern nötig.**
- **Säubern des Repositorys**
Daten aller Commits und Zweige bleiben bis zum expliziten Löschen vorhanden, um Änderungen zurücknehmen zu können.

Git Eigenschaften

- **Web-Interfaces**

Es existieren eine **Hülle von Web-Interfaces**, die als zentraler Git Server agieren können.

Die drei Hauptzustände von Git

- Git definiert **drei Hauptzustände**, in denen sich eine Datei befinden kann:
 - × **Committed**
 - × **Modified**
 - × **Staged**
- Modified bedeutet, dass eine Datei geändert, aber noch nicht in die lokale Datenbank eing检echeckt wurde.
- Staged bedeutet, dass eine geänderte Datei in ihrem gegenwärtigen Zustand für den nächsten Commit vorgemerkt ist.

Die drei Hauptzustände von Git

- Git definiert **drei Hauptzustände**, in denen sich eine Datei befinden kann:
 - × **Committed**
 - × **Modified**
 - × **Staged**
- Committed bedeutet, dass die Daten sicher in der lokalen Datenbank gespeichert sind.

Versionskontrollsystem – Git

Die drei Hauptbereiche eines Git-Projekts

Arbeitsverzeichnis
(Working Directory)

Index
(Staging Area)

Repository
(HEAD, .git Ordner)

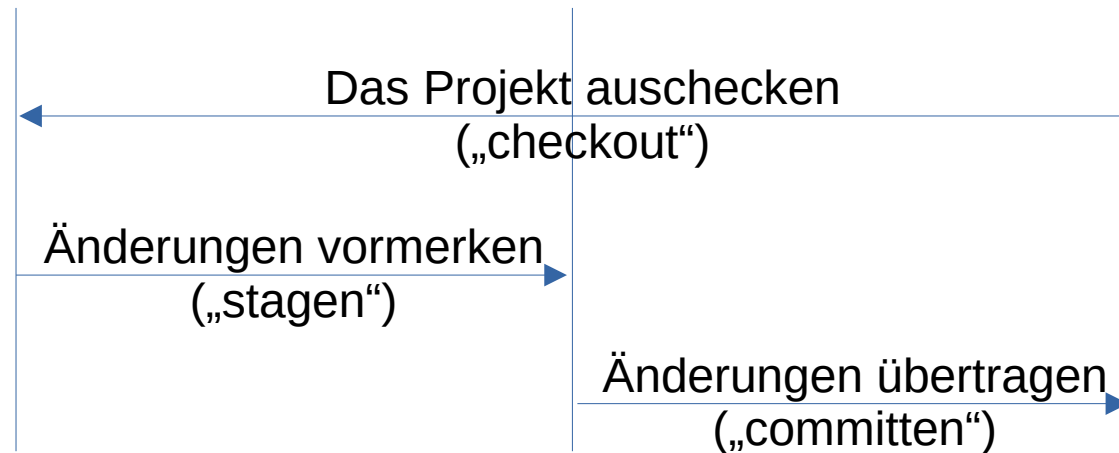


Abbildung 4: Hauptbereiche von Git
Andreas Schaffhauser, MSc.

Die drei Hauptbereiche eines Git-Projekts

Arbeitsverzeichnis (Working Directory)

Dies ist der **Bereich, in dem an den Daten gearbeitet wird**, d.h. hier befinden sich die entsprechenden Daten des Projektes. Änderungen, die hier vorgenommen werden, sind noch nicht für Git erfasst.

Index (Staging Area)

Der **Index ist eine Zwischenstufe zwischen dem Arbeitsverzeichnis und dem Repository**. Hier werden Änderungen, die vorgenommen wurden und die in das Repository aufgenommen werden, zwischengespeichert. Bevor Änderungen ins Repository übertragen werden, werden sie dem Index hinzugefügt.

Die drei Hauptbereiche eines Git-Projekts

Repository (HEAD)

Das **Repository** repräsentiert den Zustand des Projekts zu einem bestimmten Zeitpunkt. Es enthält alle Versionen der Dateien und die Historie der Änderungen. Der **HEAD** ist ein spezieller Zeiger im Repository, der auf die aktuelle Version zeigt. Wenn eine Änderung dauerhaft gespeichert werden soll, wird ein Commit durchgeführt, der die Änderungen im Repository festhält und den HEAD auf die neue Version aktualisiert.

Der grundlegende Git-Arbeitsablauf

- 1) Sie **ändern** Dateien in Ihrem Verzeichnisbaum.
- 2) Sie **stellen selektiv Änderungen bereit, die Sie bei Ihrem nächsten Commit berücksichtigen möchten**, wodurch nur diese Änderungen in den Staging-Bereich aufgenommen werden.
- 3) Sie **führen einen Commit aus, der die Dateien so übernimmt, wie sie sich in der Staging-Area befinden** und diesen Snapshot dauerhaft in Ihrem Git-Verzeichnis speichert.

Initialisierung eines leeren lokalen Git Repositorys

- Erstellen Sie einen neuen Ordner, welcher das Projekt enthalten soll.
- Navigieren Sie per Kommandozeile in den neuen Ordner.
- Geben Sie `git init` ein.
- Sie werden mit der Meldung „**Initialized empty Git repository in [path]**“ auf der Kommandozeile informiert, dass ein leeres Git Repository erstellt wurde.
- **In dem Repository ist der unsichtbare Ordner `.git` entstanden, der alle notwendigen Informationen für die Versionskontrolle enthält (z.B. Commits, Remote Repository Adresse, ...)**

Inhalt des .git Ordners

- Nach dem Ausführen des Befehls sind **vier Ordner** und **drei Dateien** im .git Ordner entstanden.
- Die vier Ordner sind
 - × hooks/: Beispiel Skripte
 - × info/: exclude Datei für ignorierte Muster
 - × objects/: alle „Objects“
 - × refs/: pointers zu den Commits
- Bei den drei Dateien handelt es sich um die config, description und die HEAD Datei.

Wichtige Dateien innerhalb eines Repository

- **.gitignore**

Ignorierte Dateien werden in einer speziellen Datei namens **.gitignore** verfolgt, die im Root-Verzeichnis Ihres Repositorys eingecheckt wird. Es gibt keinen expliziten "git ignore"-Befehl: Stattdessen muss die **.gitignore**-Datei manuell bearbeitet und committet werden, wenn neue Dateien hinzukommen, die ignoriert werden sollen.

- **README.md**

Sie können einem Repository eine **README-Datei** hinzufügen, um wichtige Informationen zu Ihrem Projekt mitzuteilen. Eine **README-Datei** kommuniziert zusammen mit einer Repository-Lizenz, einer Zitierdatei, Beitragsrichtlinien und einem Verhaltenskodex die Erwartungen an Ihr Projekt und hilft Ihnen bei der Verwaltung von Beiträgen.

Versionskontrollsystem – Git

Hinzufügen einer .gitignore und eine README.md Datei

- Navigieren Sie mit der Kommandozeile in Ihr Git Repository.
- Erstellen Sie eine .gitignore und eine README.md Datei.
- Um **beide Dateien in die Staging Area für den nächsten Commit vorzumerken**, geben Sie `git add .` im Wurzelverzeichnis des Repositorys an oder fügen jede Datei einzeln der Staging Area hinzu (`git add <dateiname>`).
- Sie können sich mit `git status` den **aktuellen Zustand Ihres Arbeitsverzeichnis und seiner Unterordner ansehen**. Somit können Sie sehen, welche Dateien noch untracked oder gestaged sind.

Commit der hinzugefügten .gitignore und README.md Datei





- Um nun die .gitignore und README.md Datei in die Datenbank des lokalen Repositorys zu übernehmen, müssen die Dateien aus der Staging Area committet werden.
- Um den Commit zu vollziehen und diesen mit einer nützlichen Commitmessage auszustatten, setzen Sie folgenden Befehl im Wurzelverzeichnis ab:
 - × `git commit -m ".gitignore und README.md added."`
- Falls Sie noch nicht Ihre E-Mail Adresse und Ihren Benutzernamen gesetzt haben, müssen Sie dies noch über die folgenden zwei Befehle tun:
 - × `git config --global user.email "you@example.com"`
 - × `git config --global user.name "Your Name"`

Hinzufügen eines Remote Repository

- Damit **das lokale Repository mit einem Remote Repository interagieren kann**, muss ein **Remote Repository angelegt werden**.
- Dafür erstellt man ein Repository z.B. unter Github.
- In dem Wurzelverzeichnis fügt man das Remote Repository mit folgendem Befehl hinzu:
 - * `git remote add origin https://github.com/username/repo.git`
- Dann benennt man den master Branch in seinem lokalen Repository zu main um und pushed das lokale Repository zum Remote Repository.
 - * `git branch -M main`
 - * `git push -u origin main`

Überprüfen des Commits

- Sie können den stattgefundenen Commit über Ihren Account auf dem jeweiligen Webinterface einsehen.

 fsfi	.gitignore und README.md added.	cb9dd15	12 minutes ago	 1 commit
	.gitignore	.gitignore und README.md added.	12 minutes ago	
	README.md	.gitignore und README.md added.	12 minutes ago	

.gitignore und README.md added.
 fsfi committed 13 minutes ago

 [cb9dd15](#) 

Versionskontrollsystem – Git

.gitignore Datei Muster ^[3]

- **.gitignore-Dateien enthalten Muster**, die mit den Dateinamen in deinem Repository abgeglichen werden, um zu bestimmen, ob diese ignoriert werden sollen oder nicht.
- **.gitignore nutzt Globbing-Muster zur Abgleichung der Dateinamen**. Muster können mithilfe verschiedener Symbole erstellt werden:

Muster	Beispiele für Übereinstimmungen	Erklärung
**/logs	logs/ablauf.log logs/heute/bla.bar build/logs/ablauf.log	Um Verzeichnisse im gesamten Repository abzugleichen, füge einem Muster ein doppeltes Sternsymbol voran.

.gitignore Datei Muster

Muster	Beispiele für Übereinstimmungen	Erklärung
**/logs/ablauf.log	logs/ablauf.log build/logs/ablauf.log <i>aber nicht</i> logs/build/ablauf.log	Ebenso ist es möglich, Dateien anhand ihres Namens und des Namens ihrer übergeordneten Verzeichnisse durch die Verwendung von zwei Sternsymbolen abzugleichen.
*.log	ablauf.log bla.log .log logs/ablauf.log	Ein Sternsymbol ist ein Platzhalter, die mit null oder mehr Zeichen übereinstimmen kann.

.gitignore Datei Muster

Muster	Beispiele für Übereinstimmungen	Erklärung
*.log !wichtig.log	ablauf.log mitschnitt.log <i>aber nicht</i> wichtig.log logs/wichtig.log	Das Voranstellen eines Ausrufezeichens zu einem Muster negiert es. Wenn eine Datei zu einem Muster passt, aber auch zu einem später in der Datei definierten negierenden Muster passt, wird sie nicht ignoriert.
.log !wichtig/.log mitschnitt.*	ablauf.log wichtig/mitschnitt.log <i>aber nicht</i> wichtig/ablauf.log	Die Definition von Mustern nach einem negierenden Muster bewirkt, dass zuvor ausgeschlossene Dateien erneut ignoriert werden.

.gitignore Datei Muster

Muster	Beispiele für Übereinstimmungen	Erklärung
/ablauf.log	ablauf.log <i>aber nicht</i> logs/ablauf.log	Durch das Voranstellen eines Schrägstrichs werden nur Dateien im Hauptverzeichnis (Root-Verzeichnis) des Repositorys abgeglichen.

Persönliche Regeln für Git ignore

- Außerdem können Sie **in einer eigenen Datei unter `.git/info/exclude` persönliche ignore-Muster** für ein bestimmtes Repository definieren.
- **Diese sind nicht versioniert und werden nicht mit Ihrem Repository verteilt**, sodass hier Muster enthalten sein können, die ausschließlich für Sie nützlich sind.
- Wenn Sie beispielsweise individuelle Protokollierungseinstellungen verwenden oder spezielle Entwicklertools nutzen, die Dateien im Arbeitsverzeichnis Ihres Repositories erstellen, besteht die Möglichkeit, sie der Datei `.git/info/exclude` hinzuzufügen. Dadurch wird vermieden, dass sie versehentlich in das Repository committet werden.

README.md

- Die **Erweiterung .md** kommt von dem Wort **Markdown**.
- Es ist eine **Auszeichnungssprache für die Textformatierung**. Ein Ziel von Markdown ist eine leicht lesbare Ausgangsform bereits vor der Konvertierung.
- Inhalte, die eine gute README .md enthalten sollte
 - × Titel und Einführung in die Software
 - × Verwendete Technologien
 - × Voraussetzungen zum Starten der Software
 - × Anwendungsbeispiel

Titel und Einführung

- Ein **Titel sollte klar erklären, was wir haben**, und es ist **normalerweise ein Projektname** - eine H1-Überschrift mit vorangestelltem #.
- Überschriften werden folgendermaßen in verschiedenen Ebenen realisiert:
 - x # header H1
 - x ## header H2
 - x ### header H3
 - x #### header H4
 - x ##### header H5
 - x ##### header H6

Titel und Einführung

- Die **Einführung ist ähnlich einer Zusammenfassung**.
- Es sollte **nicht zu ausführlich** sein, **da wir keinen ausführlichen Bericht über das Projekt lesen möchten**. Es ist besser, **kurz und prägnant zu beschreiben**, was das Projektziel ist und welche Probleme durch eine bestimmte Anwendung gelöst werden. Für kleinere Projekte genügen oft zwei bis drei Sätze.
- Z.B. kann es im Rahmen von einer Vorlesung als Übung gedient haben. Erklären Sie hierbei den Kontext der Vorlesung und den Zweck der Übung.

Verwendete Technologien

- **Listen Sie die verwendeten Technologien und Sprachen auf**, die Sie für Ihr Projekt genutzt haben.
- Gute Gründe hierfür sind:
 - × Welche **Technologieversionen** wurden genutzt als unser Projekt noch problemlos lief?
 - × **IT-Rekrutierung**: Rekruter*innen surfen durch die GitHub Accounts von potenziellen Bewerbern und suchen nach Schlagwörtern, z.B. für bestimmte Technologien.

Anwendungsbeispiel

```
...  
python example.py --help  
Usage: example.py [options]  
Options:  
-h, --help                Show this help message and exit  
-f FILE, --file=FILE      Specify the file (PNG-format) that you want to check  
-c CSV, --csv=CSV         Write the output log to a csv file: default is to wrote on the  
shell  
...
```

Inhaltsverzeichnis

Table of contents

- * [General info](#general-info)
- * [Technologies](#technologies)
- * [Setup](#setup)

General info

This project is simple Lorem ipsum dolor generator.

Technologies

Project is created with:

- * Lorem version: 12.3

...

Literatur

[1]: Vgl. <https://de.wikipedia.org/wiki/Git>

[2]: Vgl. https://de.wikipedia.org/wiki/Linus_Torvalds

[3]: Vgl. <https://git-scm.com/book/de/v2>