

ISW: Software Engineering

WS 2023/24

Einführung in Versionsverwaltung mit Git

Michael Anders

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 205
69120 Heidelberg, Germany
<https://se.ifi.uni-heidelberg.de>



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

1. Probleme während der Softwareentwicklung
2. Lösung: Versionsverwaltung
3. Erste Schritte mit Git
4. Installation und Konfiguration von Git
5. Projekt importieren und freigeben
6. Änderungen einpflegen
7. Fortgeschrittene Technik in Git
8. Git-Internia
9. Webanwendungen für Git

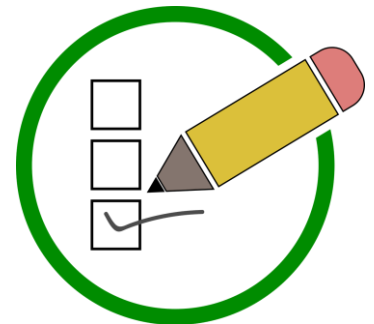
Wie viel Erfahrung haben Sie mit Git?

A = benutze ich ständig

B = benutze ich ab und zu

C = habe ich selten genutzt

D = habe ich noch nie genutzt



Probleme während der Entwicklung (1/3)

■ Softwareentwicklung ist nicht linear

- Es wurden Fehler in der Implementierung gemacht und diese möchte man korrigieren
- Getroffene Entscheidungen sind nicht immer die besten

■ Software wird im Team entwickelt

- Viele EntwicklerInnen arbeiten gemeinsam in einem Projekt
- Die EntwicklerInnen wollen wissen, wer etwas im Projekt getan hat (was, wann, warum)
- Der/die einzelneN EntwicklerIn möchte wissen, was er/sie getan hat (wann, warum)

■ Wartung von unterschiedlichen Versionen

- KundenInnen bekommen stabile Version (Release)
- Die interne Entwicklung geht aber weiter
- Bug-Fixes müssen in die stabile Version eingepflegt werden

Probleme während der Entwicklung (2/3)

- **Backups sind notwendig, aber:**
 - man verliert oft den Überblick
 - man weiß nicht, welches Backup von welcher Version erstellt wurde
 - oder in welcher Version der Bug X korrigiert wurde
 - komplette (oder iterative) Backups benötigen viel Zeit und Speicherplatz
- **80/20 Regel: 20% Entwicklung, 80% Debugging**
 - “Ich kann den Bug nicht finden ...” / “Gestern hat es noch funktioniert ...”
- **Bei Wechsel des Arbeitsbereichs muss Projekt kopiert werden**
 - Inkonsistenzen sind vorprogrammiert
- **Kommunikation wird vergessen**
 - “In Klasse X habe ich Y implementiert wegen Z ...”
 - “Der Bug X in Klasse Y ist korrigiert ...”
 - “Stop! Die Version ist fehlerhaft! Ich gebe dir meine ...”

Probleme während der Entwicklung (3/3)



Wo ist der Unterschied?

Aktuelle Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething1();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

Vorherige Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething2();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

Aktuelle Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething1();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

Vorherige Version

```
public void myTestMethod()  
{  
    log("Test method entered.");  
  
    boolean keepOnRunning = true;  
    int i=0;  
  
    while( keepOnRunning )  
    {  
        int r1 = doSomething1();  
        int r2 = doSomething2();  
  
        if( r1+r2 > MAX_THRESHOLD )  
        {  
            log("Threshold exceeded.");  
            log("Iterations: " + i);  
            keepOnRunning = false;  
        }  
        else  
        {  
            printStatus(r1, r2);  
            i++;  
        }  
    }  
}
```

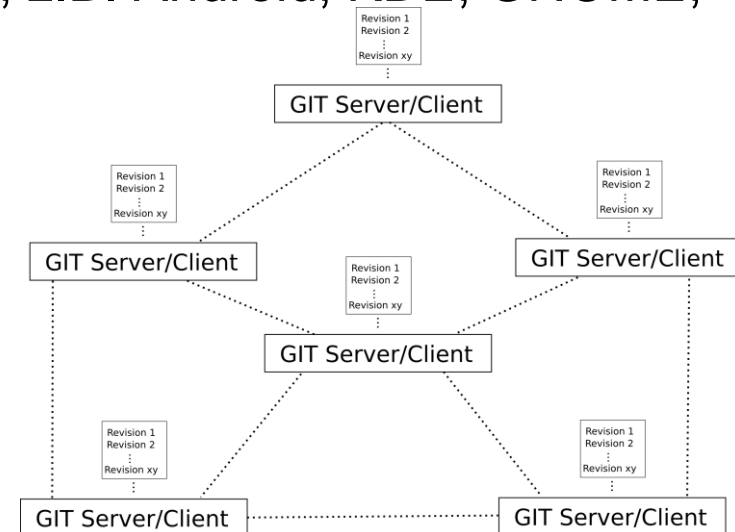

Was kann Versionsverwaltung leisten?

- Sichern der eigenen Arbeit
- Änderungen rückgängig machen
- Mit anderen Personen kollaborativ zusammenarbeiten
 - Mehrere Personen können Dokumente teilen und editieren
- Dokumente sind online verfügbar
 - Wenn das Repository online erreichbar ist, hat man von überall Zugriff auf die Dateien

- Beginn: April 2005 als Ersatz für BitKeeper
- Wird von sehr vielen Projekten verwendet, z.B. Android, KDE, GNOME, LibreOffice, Linux-Kernel, Eclipse
- Dezentralisiertes Client-Server System

- Clients:

- Command Line (<http://git-scm.com/downloads>)
- SourceTree, GitEye, GitKraken, ...
- IDE-Clients:
 - Git-Plugin für Android Studio (<https://plugins.jetbrains.com/plugin/13173-git>)
 - EGit (Plugin für Eclipse) (<http://www.eclipse.org/egit/>)



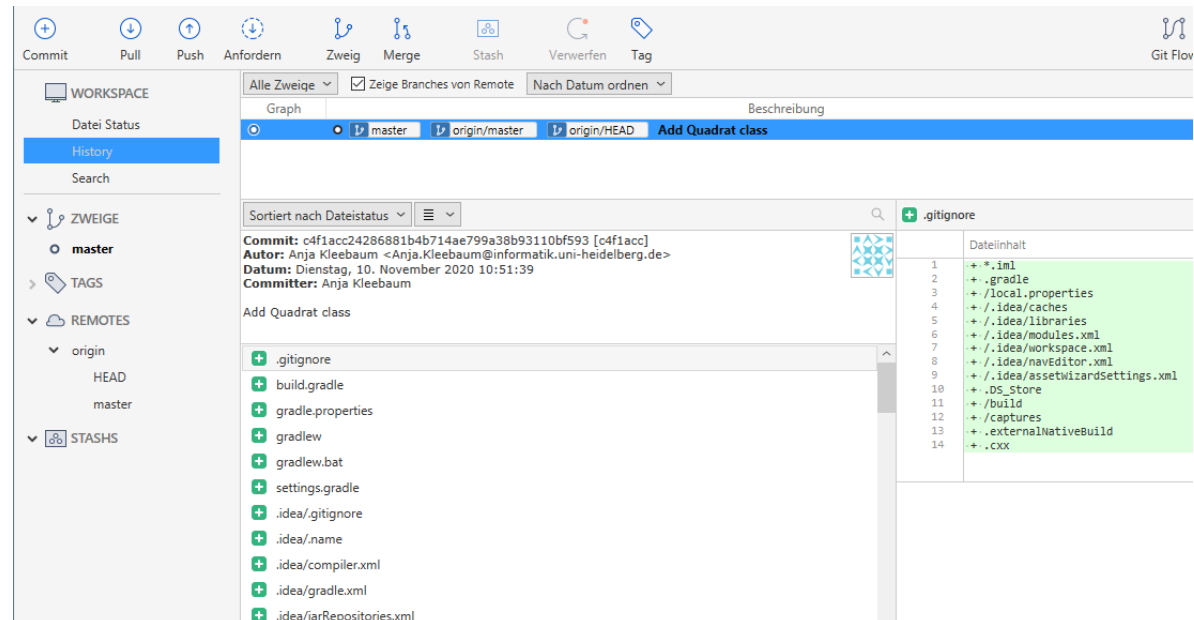
- **(Meist) trotzdem ein zentrales Repository (Remote Repository)**
 - “öffentliches” Repository

- **Alle Unterschiede zwischen zwei oder mehr Versionen sind erkennbar**
 - “Welche Änderungen gibt es in der heutigen Version im Vergleich zur gestrigen Version?”
 - “Was haben andere Entwickler in Klasse X geändert?”

- **Erlaubt Änderungen (eigene oder fremde) zu**
 - untersuchen, übernehmen, ablehnen, diskutieren

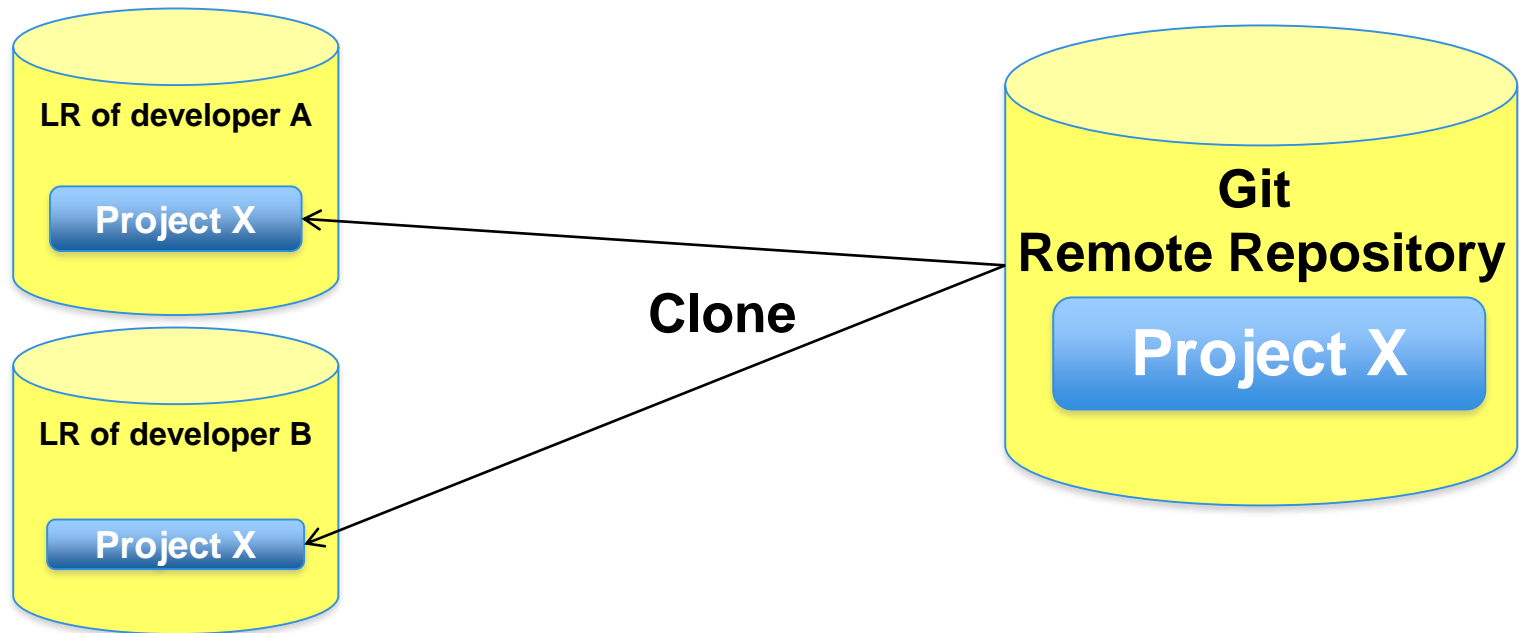
- **Vorheriger Zustand kann wiederhergestellt werden**
 - Komplett oder nur teilweise
- **Versionen können durch Tags gekennzeichnet werden**
 - Benutzung für Releases, Meilensteine oder einfache Backups
- **Meta-Informationen**
 - Wer hat Änderungen gemacht? (wann, was, warum, wo)

- **Clone:** Kopiert ein bestehendes Git-Repository
- **Add:** Dateien zum lokalen Repository hinzufügen
- **Commit:** Änderungen im lokalen Repository veröffentlichen
- **Push:** Lokale Änderungen im Remote Repository veröffentlichen
- **Pull:** Änderungen aus dem Remote Repository übernehmen
- Repository Layout:



Clone: Initialer Download des Repository

- EntwicklerInnen “clonen” das Projekt-Repository
- Erhalten eine lokale Kopie (local repository = LR)
- Nun können sie mit ihrer Arbeit am Projekt beginnen
- “Clonen” passiert nur einmal zu Beginn → Danach nur noch “Pull”



Add + Commit: Änderungen im LR speichern

- Nach “clone” ändert EntwicklerIn das Projekt lokal
- Änderungen müssen mit “add” hinzugefügt werden

- Durch “commit” werden die Änderungen im LR veröffentlicht
- Vor einem Commit muss **immer**:
 - die eigene Änderungen überprüft werden
 - sichergestellt werden, dass die Software kompilierbar ist und Tests durchlaufen (sonst wird u.U. das gesamte Team behindert)

- Commits sind “Snapshots” der geänderten Dateien
 - Git speichert immer komplette Datei, die geändert wurde
 - Ein Commit kann mehrere geänderte Dateien umfassen

- Jeder Commit hat eine “Commit Message”, welche beschreibt, was für Änderungen gemacht wurden
- Eine “Commit Message” muss folgende Informationen enthalten:
 - Was wurde genau geändert?
 - Bezug zu Issue im Issue Tracking System
- Zu Beginn Verb im Infinitiv (z.B. „Add“ anstatt „added“)
- Nach der ersten Zeile folgt eine Leerzeile

So nicht...:

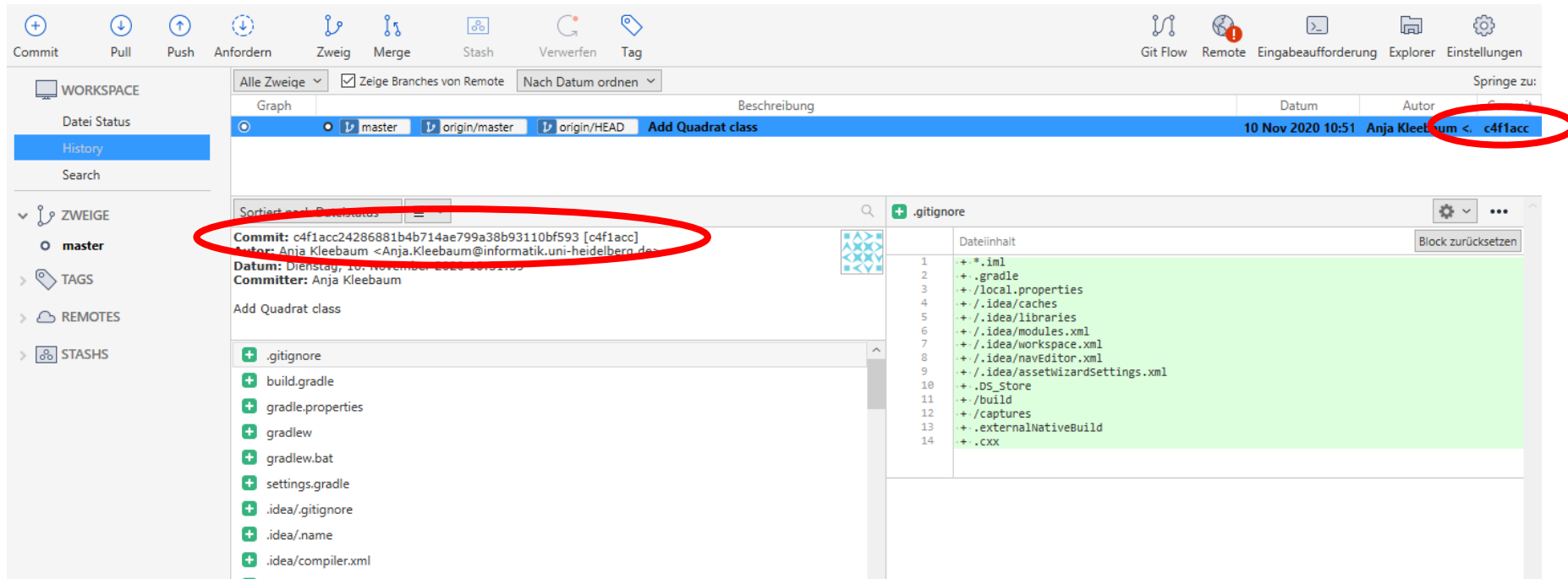
	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

- <https://chris.beams.io/posts/git-commit/>

Revisionsnummer (Commit-ID)

- Automatisch generierte eindeutige Identifikationsnummer
- Wird pro Commit erzeugt
- Bezieht sich auf alle Dateien innerhalb des Commits



The screenshot shows the IntelliJ IDEA interface with the commit history panel open. The commit history table at the top shows a single commit with the message "Add Quadrat class", dated "10 Nov 2020 10:51", by "Anja Kleebaum", and with the commit ID "c4f1acc". The commit ID is circled in red. Below the table, the details of the selected commit are shown, including the commit message "Add Quadrat class", the author "Anja Kleebaum", the date "Dienstag, 10. November 2020 10:51", and the committer "Anja Kleebaum". The commit ID "c4f1acc" is also circled in red. The file list on the right shows the files included in the commit, including ".gitignore", "build.gradle", "gradle.properties", "gradlew", "gradlew.bat", "settings.gradle", ".idea/.gitignore", ".idea/.name", and ".idea/compiler.xml".

Push & Pull: Änderungen an RR übertragen und von RR übernehmen

- Push: veröffentlicht mindestens einen Commit in dem RR
- Pull: Der aktuelle Stand im RR wird in die lokale Kopie (LR) übernommen
- Nach längerer Pause IMMER Projekt aktualisieren BEVOR mit der Arbeit fortgefahren wird!
- Änderungen werden angezeigt und können mit “Add” übernommen werden

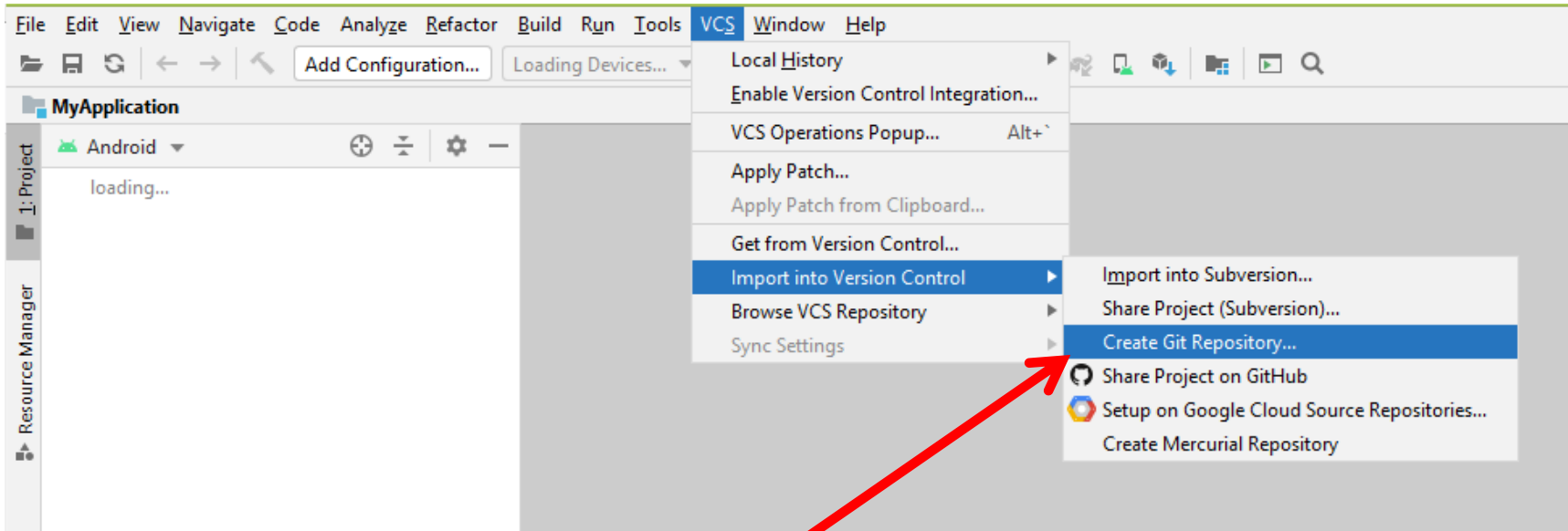
Installation und Konfiguration von Git

- Git-Installation auf Rechner <https://git-scm.com/downloads>
- Git-Client-Plugin für Android Studio is bereits enthalten
 - Stellt die Funktionalität von Git in der Android Studio IDE bereit
- Git-Konfiguration:

```
$ git config --global user.name "John Doe"
```

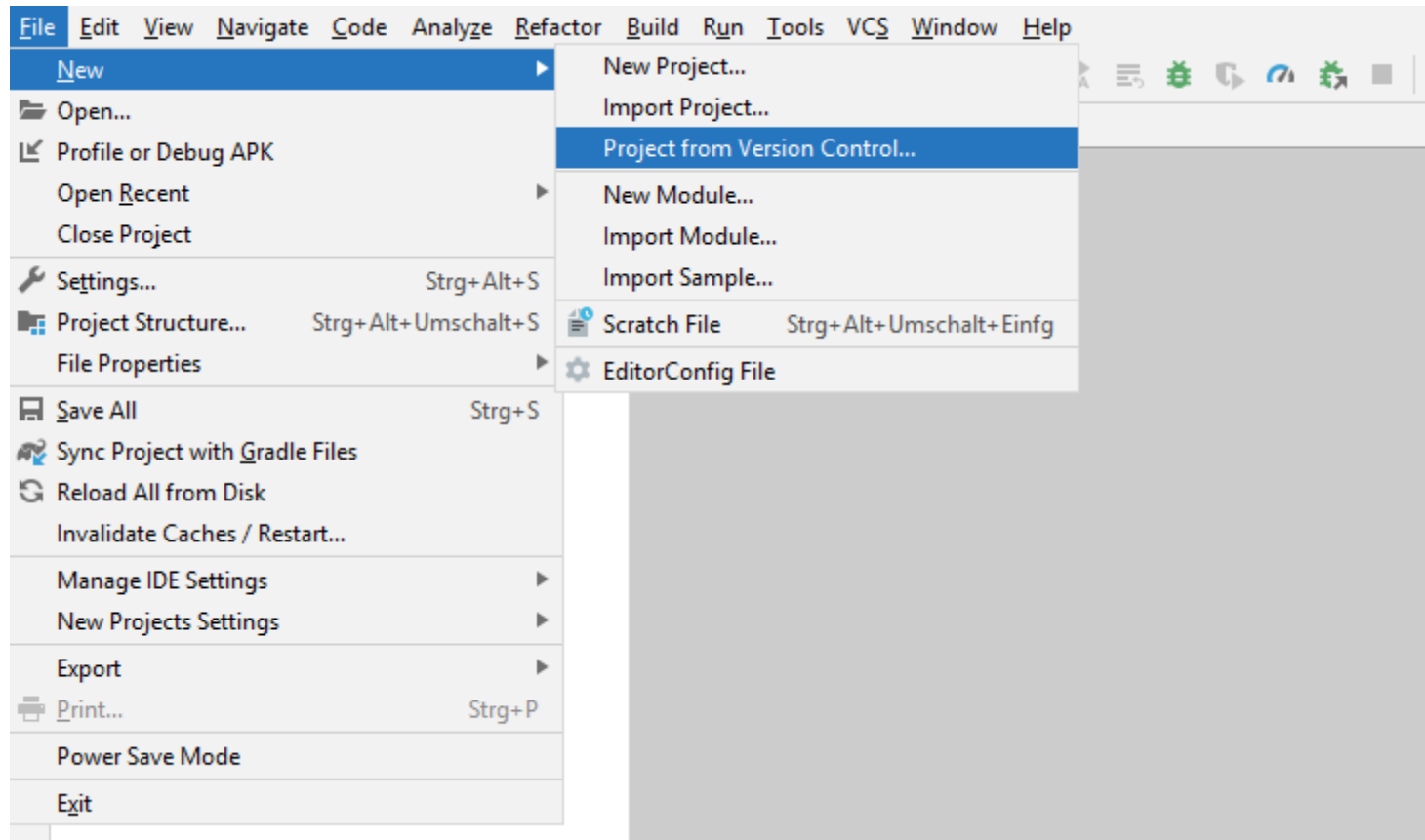
```
$ git config --global user.email johndoe@example.com
```

Lokales Repository erstellen

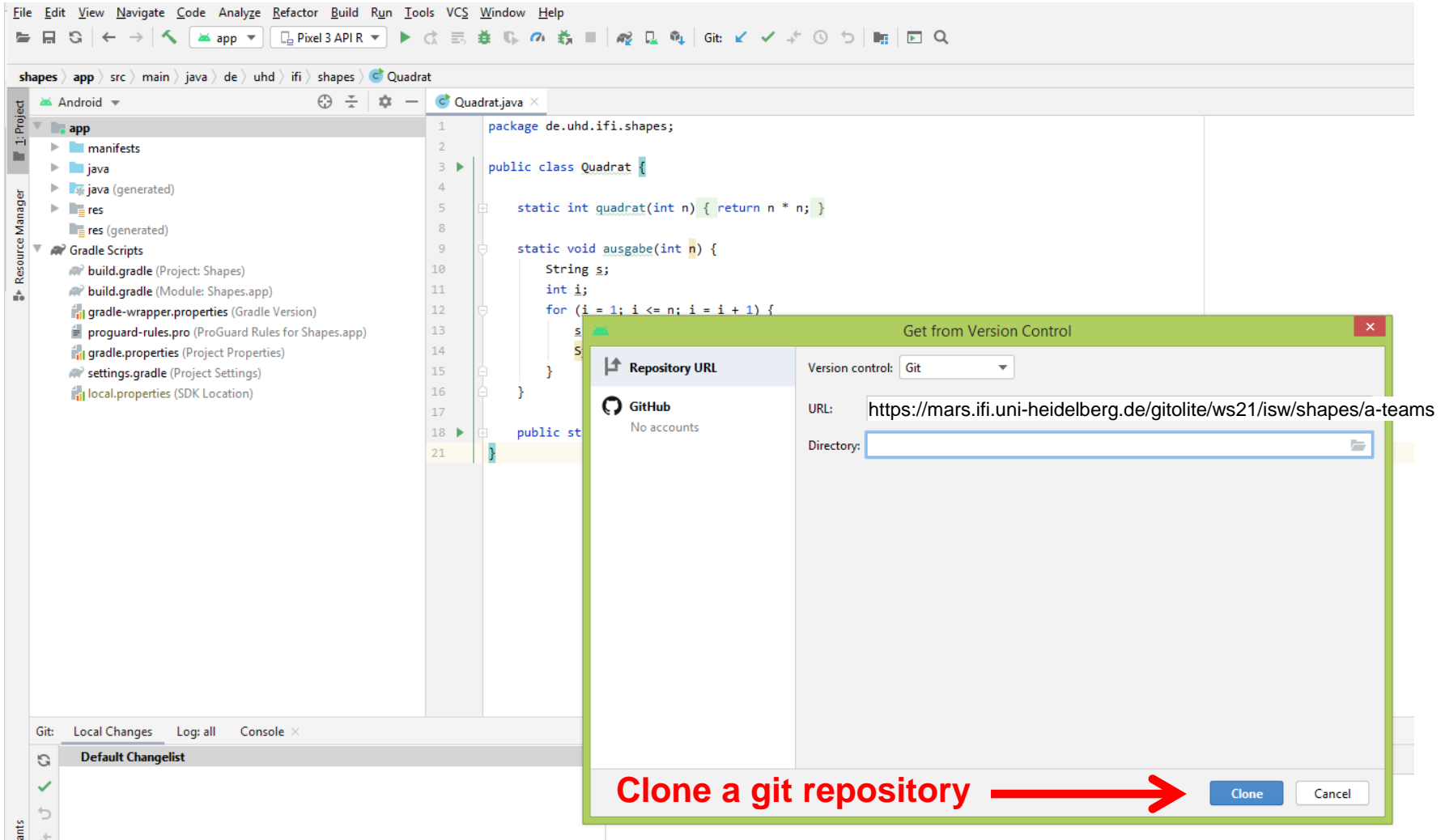


Create a new Git Repository

Repository clonen (1/2)



Repository clonen (2/2)



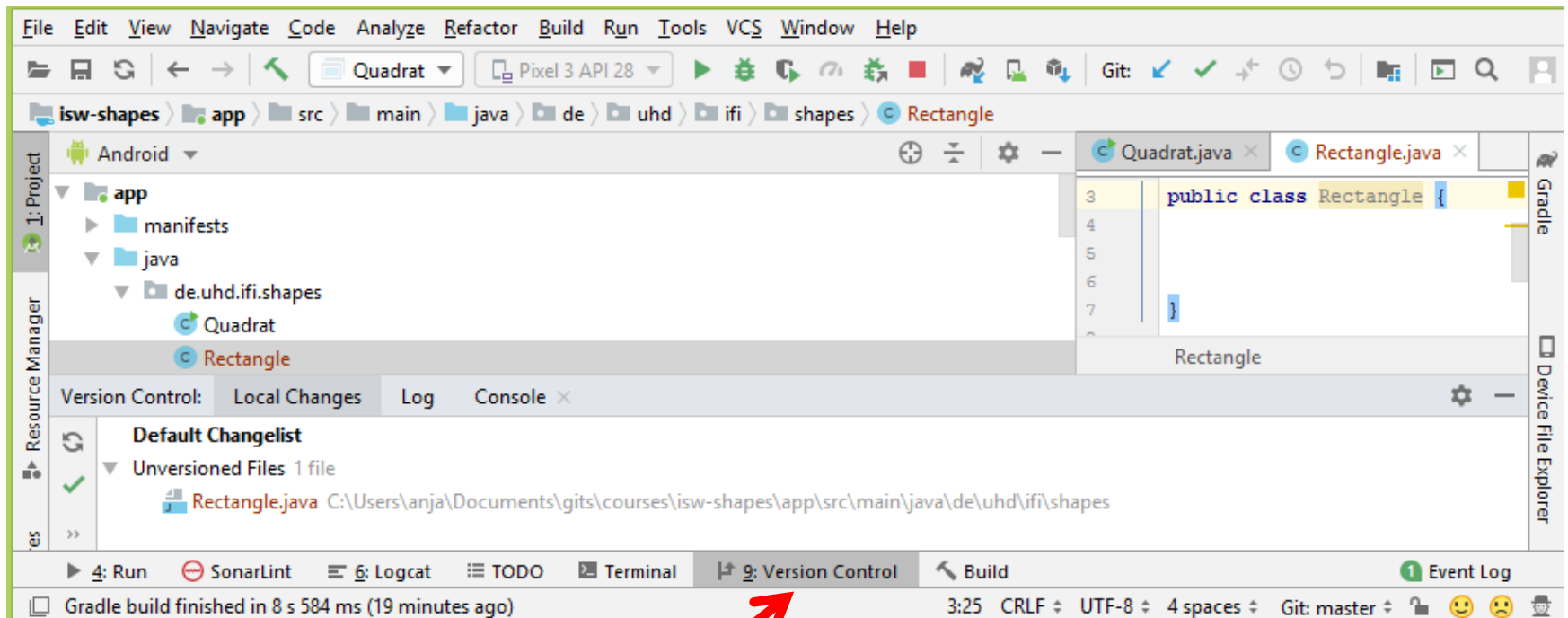
The screenshot shows an IDE window with a project named 'shapes' and a file named 'Quadrat.java'. The code in 'Quadrat.java' is as follows:

```
1 package de.uhd.ifi.shapes;
2
3 public class Quadrat {
4
5     static int quadrat(int n) { return n * n; }
6
7
8     static void ausgabe(int n) {
9         String s;
10        int i;
11        for (i = 1; i <= n; i = i + 1) {
12            S
13        }
14    }
15
16 }
17
18 public st
19
20
21 }
```

The 'Get from Version Control' dialog box is open, showing the 'Repository URL' field with the value 'https://mars.ifi.uni-heidelberg.de/gitolite/ws21/isw/shapes/a-teams'. The 'Version control' dropdown is set to 'Git'. The 'Directory' field is empty. A red arrow points to the 'Clone' button.

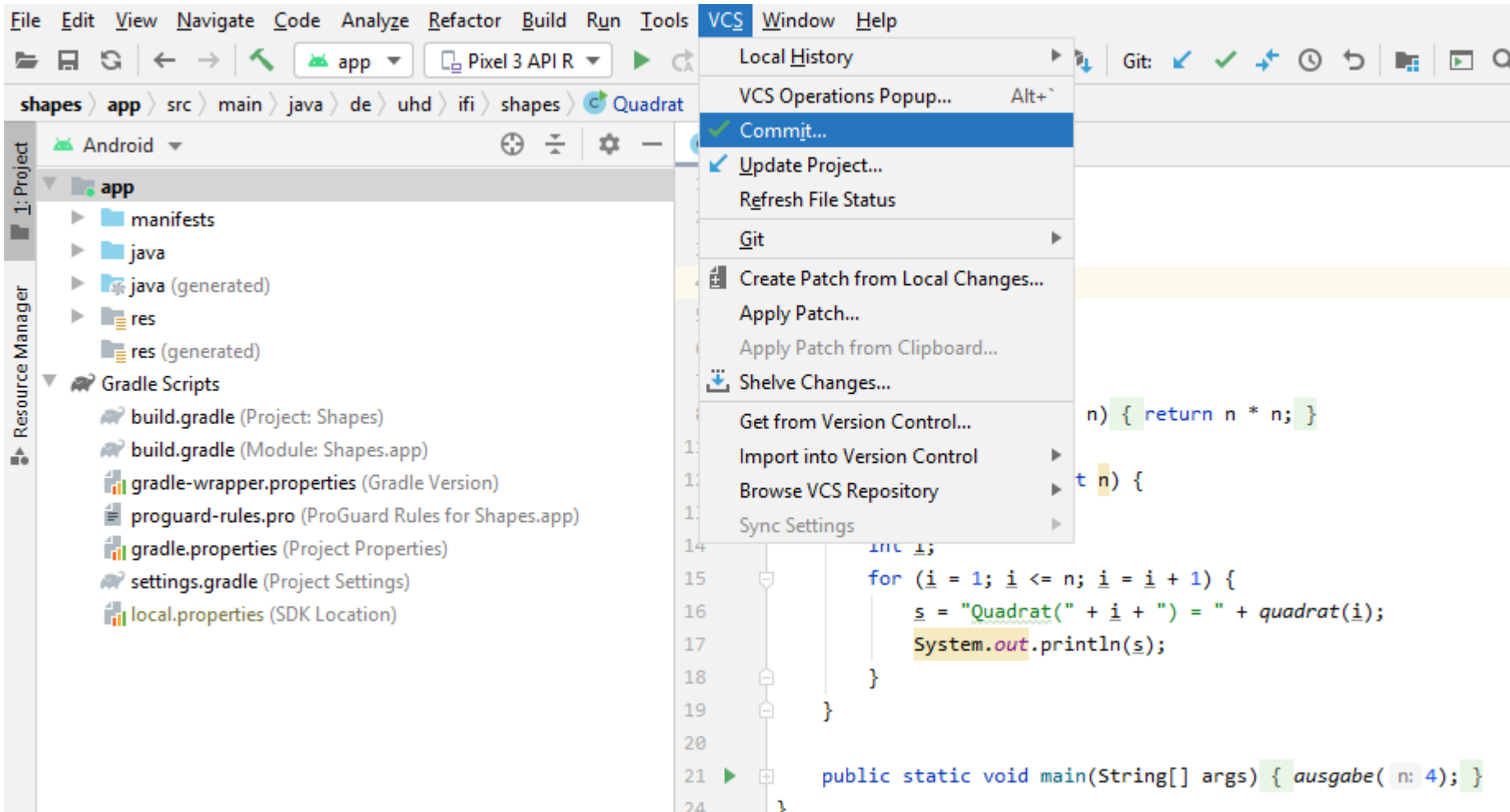
Clone a git repository →

Änderungen einpflegen (1/3)

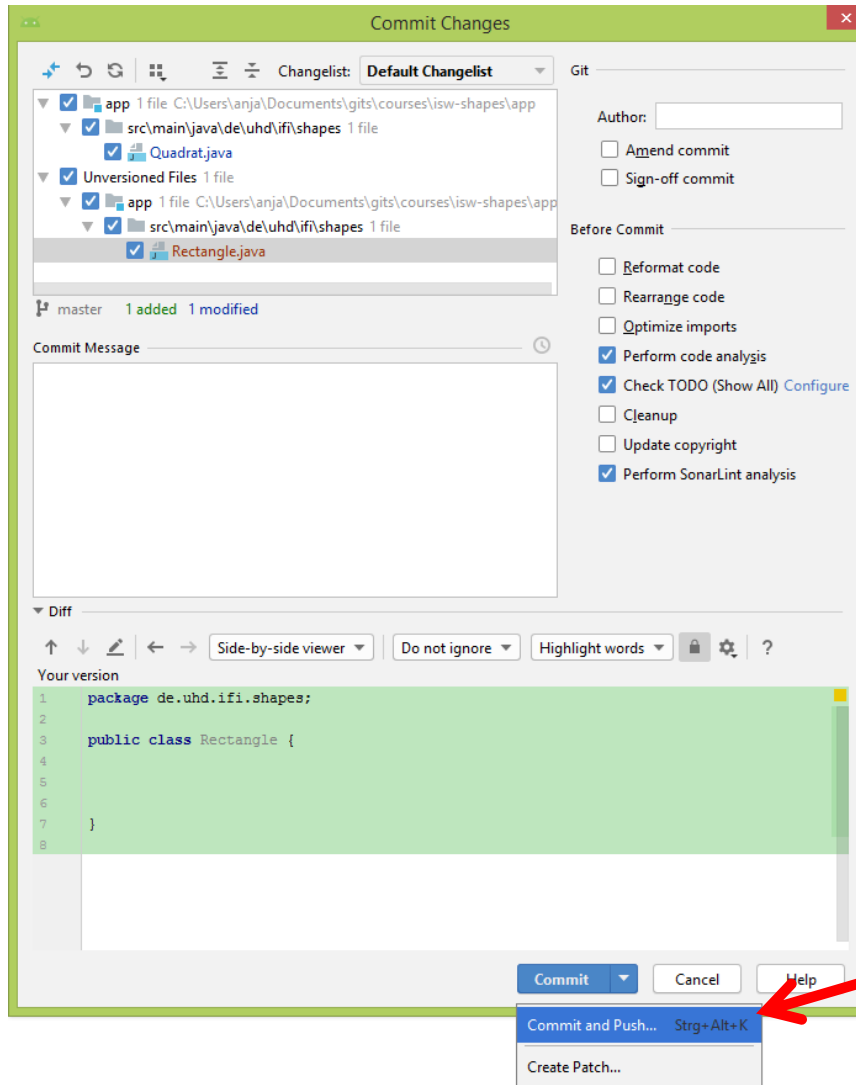


Version Control/Git Tool Window

Änderungen einpflegen (2/3)

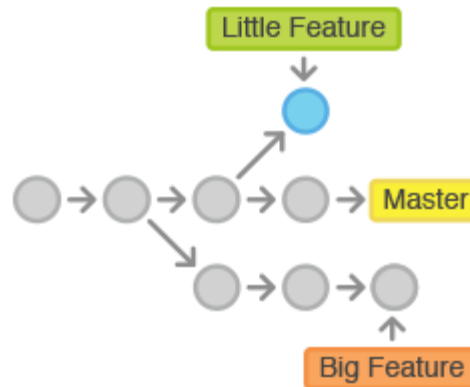


Änderungen einpflegen (3/3)



Commit and Push

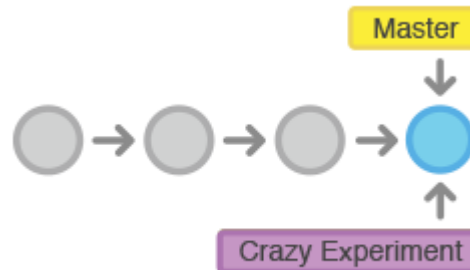
- Neue Funktion hinzufügen oder einen Fehler beheben (egal, wie groß oder klein): **neuen Branch anlegen!**
- Abkapseln von Änderungen
 - Sicherstellen, dass der instabile Code aus diesem Branch nicht zur offiziellen Codebasis hinzugefügt wird



- Branch erstellen



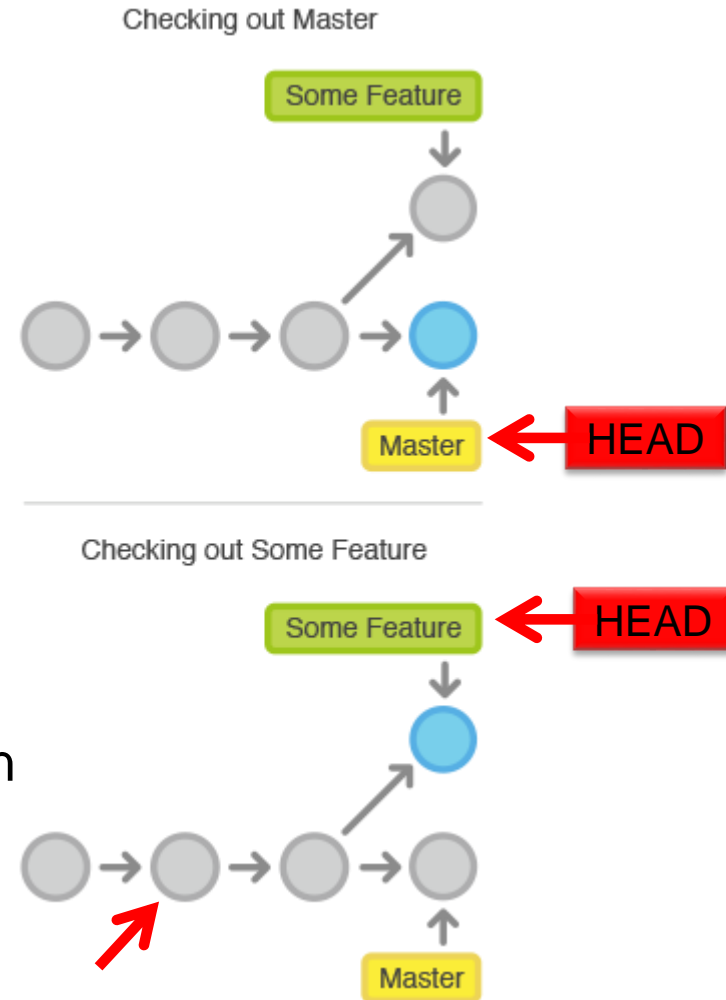
- “git branch crazy-experiment”



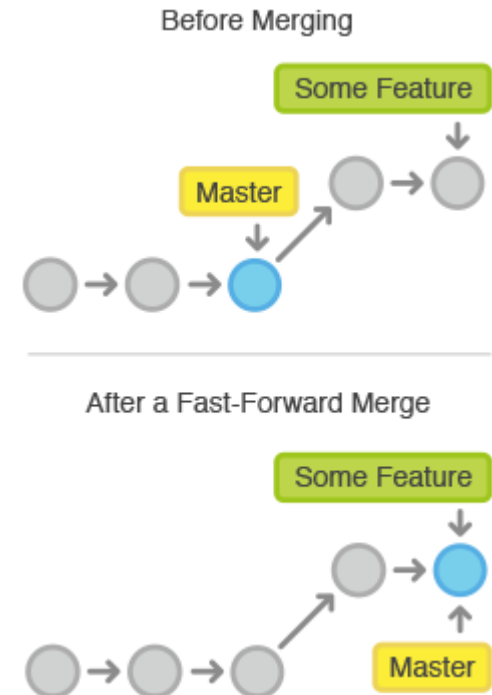
- Branches sind Zeiger (Pointer, Refs) auf Commits

- Nach dem Erstellen eines Branches muss dieser “ausgecheckt” werden
- `git checkout Some Feature`
- HEAD kennzeichnet Branch, auf dem man sich aktuell befindet
 - HEAD kann auch “detached” sein, dann ist einzelner Commit “ausgecheckt”
 - `git checkout <<Commit-ID>>`

Checkout von Commit führt zu detached HEAD



- Nach Abschluss der Entwicklung muss der Branch mit dem Hauptzweig zusammengeführt werden
- Mit dem Befehl „git merge“ lassen sich unabhängige Entwicklungszweige, die mit „git branch“ erstellt wurden, wieder in einen einzelnen Branch zusammenführen



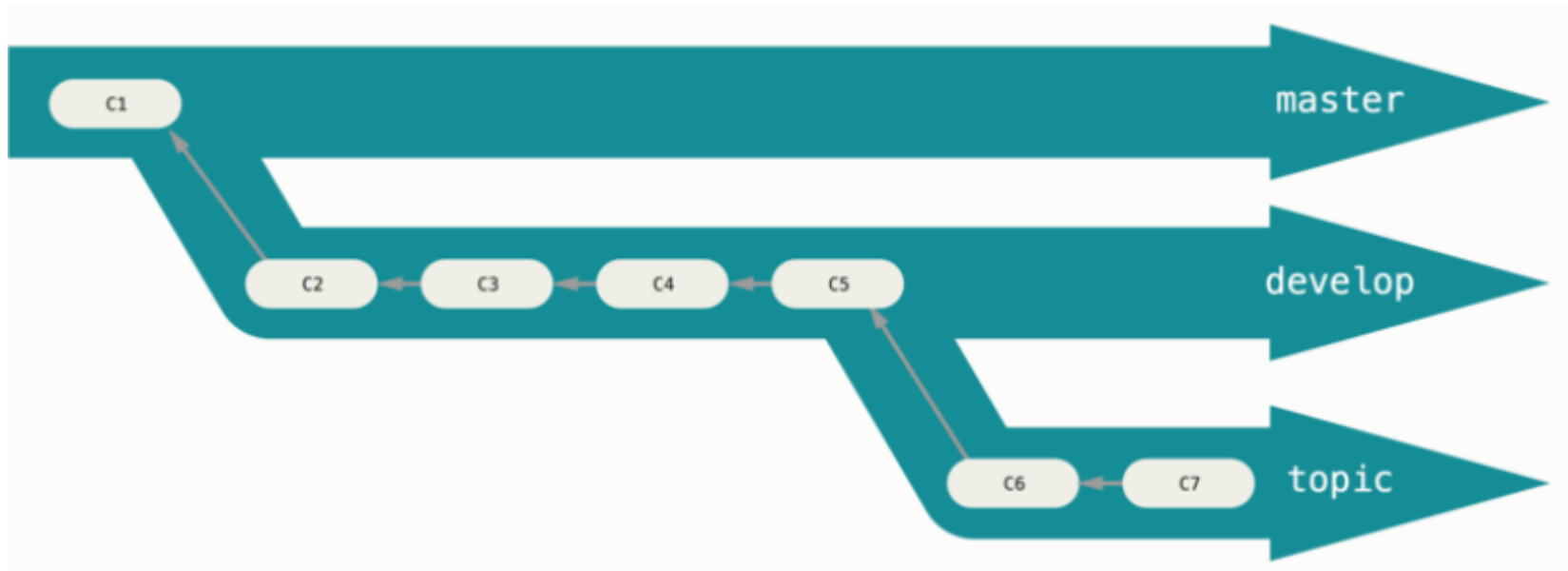
- Datei wurde auf Branches, die Sie zusammenführen wollen, an derselben Stelle unterschiedlich geändert

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

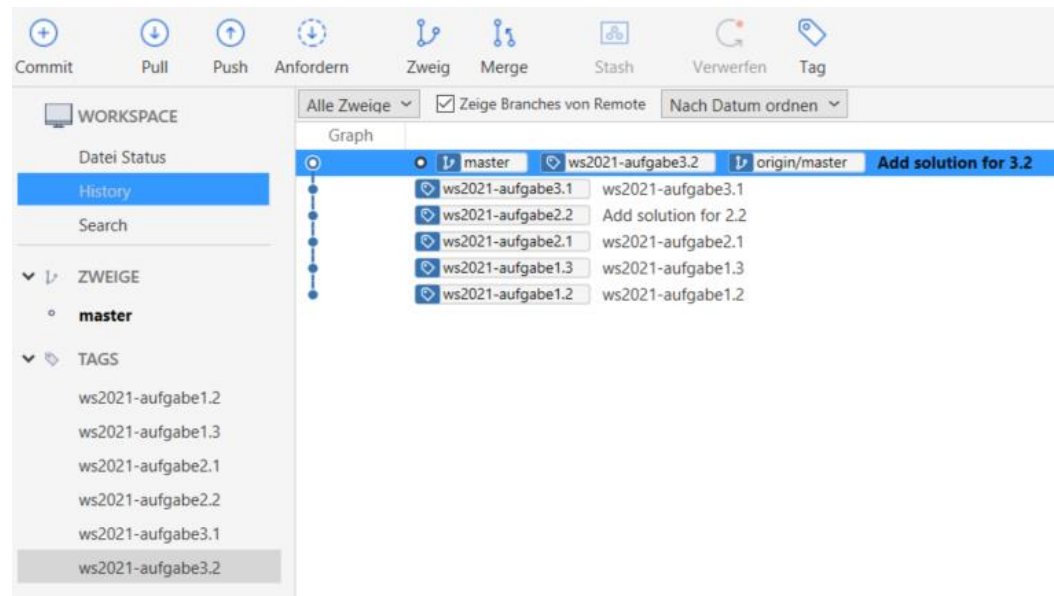
- Datei muss manuell geöffnet und eine Variante gewählt werden

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

- Langfristige Branches: master, develop
- Themen-Banches (topic branch, feature branch) sind kurzlebige Branches



- Tags markieren bestimmte Punkte in der Historie eines Repositorys als wichtig
 - Releases (Versions-Tags) z.B. v1.0
- Zeiger (Refs), die auf Punkte im Git-Verlauf verweisen
- Tag ähnelt Branch, der sich nicht ändert



Name

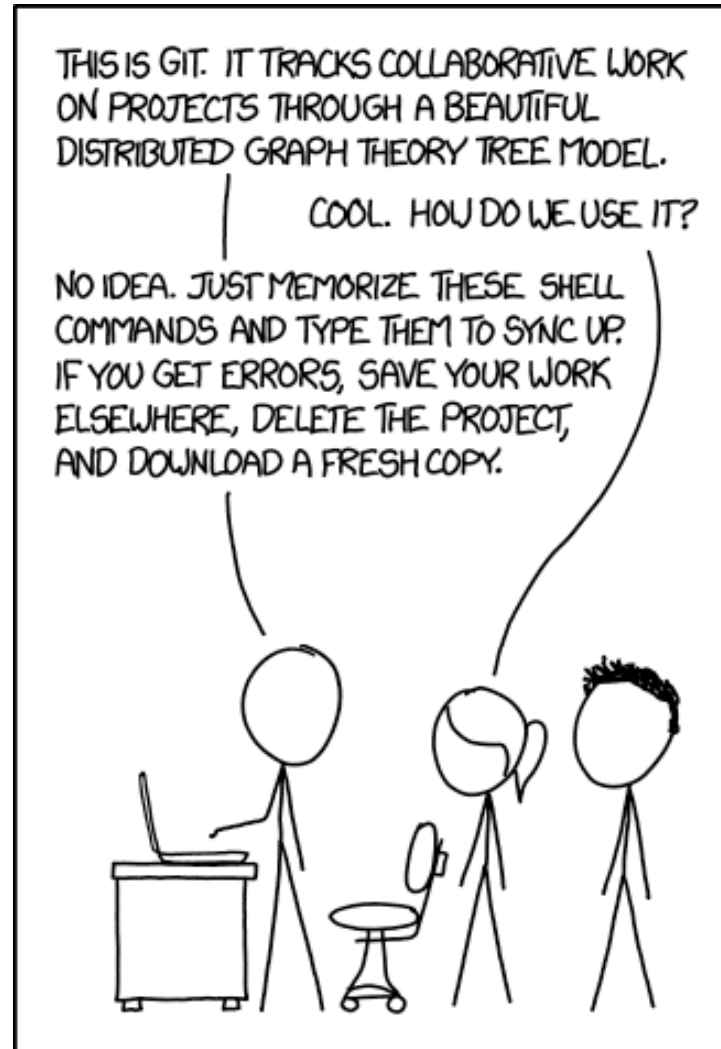
git add
git clone
git commit
git push
git add -u
git diff
git help
git log
git merge
git revert
git reset
git status
git pull
git branch
git checkout

Purpose

Add files and/or directories to version control.
Get a fresh working copy of a remote repository.
Commit changes in the local repository.
Update the remote repository.
Delete files and/or directories from version control.
Shows changes for directories/files in a unified diff format.
Get help (in general, or for a particular command).
Show history of recent changes.
Merge two different versions of a file into one.
Undo pushed changes (i.e., resynchronize with remote repository).
Undo committed local changes.
Show the status of files and directories in the working copy.
Get changes from the remote repository into local repository.
Create branch based on HEAD.
Switch to a branch.

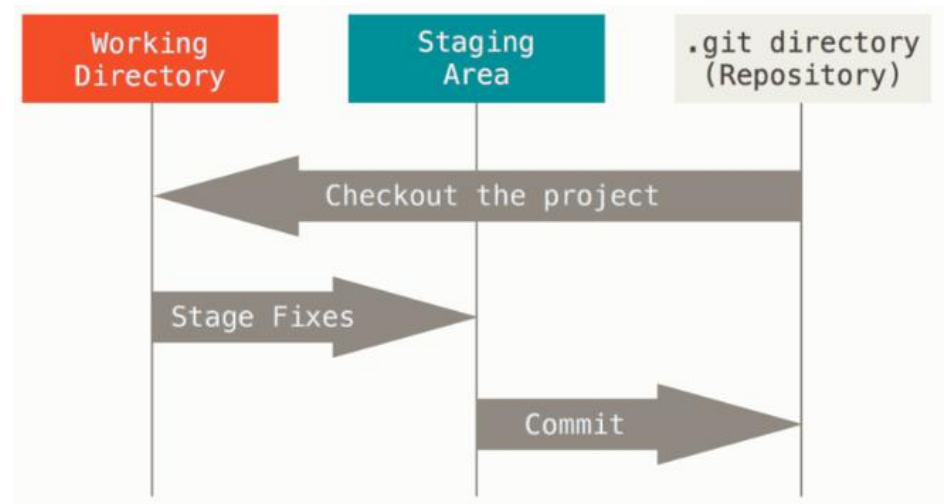
**Es gibt etwa 30
Standardbefehle
(porcelain), darunter
Basisbefehle
(plumbing)**

Git Command Reference: So nicht...



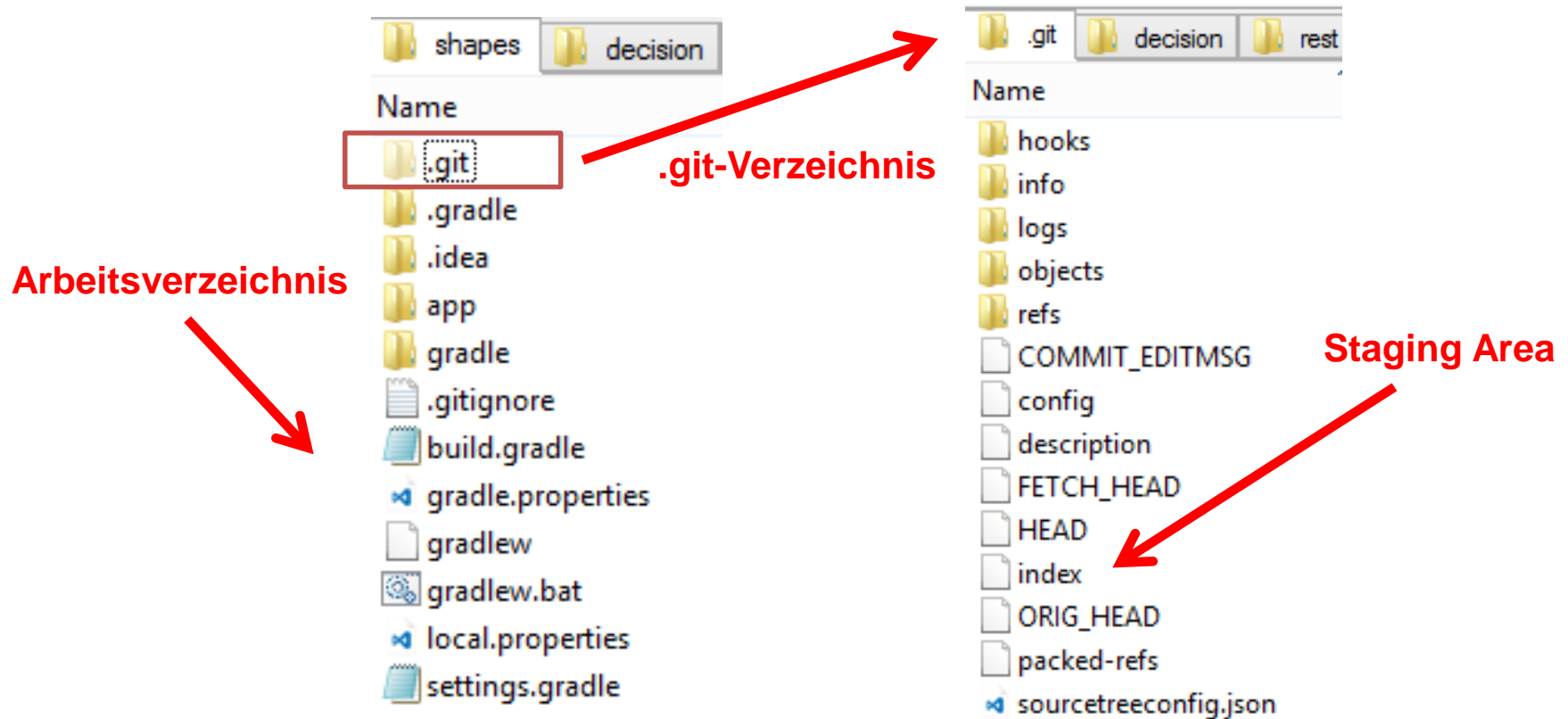
Git-Interna: 3 Hauptbereiche

- Dateien im **Arbeitsverzeichnis** sind entweder
 - tracked (durch git add) oder
 - untracked
- **Staging Area** enthält geänderte Dateien (tracked, modified und staged), die in nächsten Commit einfließen
- Commits und Historie werden in **.git-Ordner** gespeichert (ähnlich zu Datenbank)



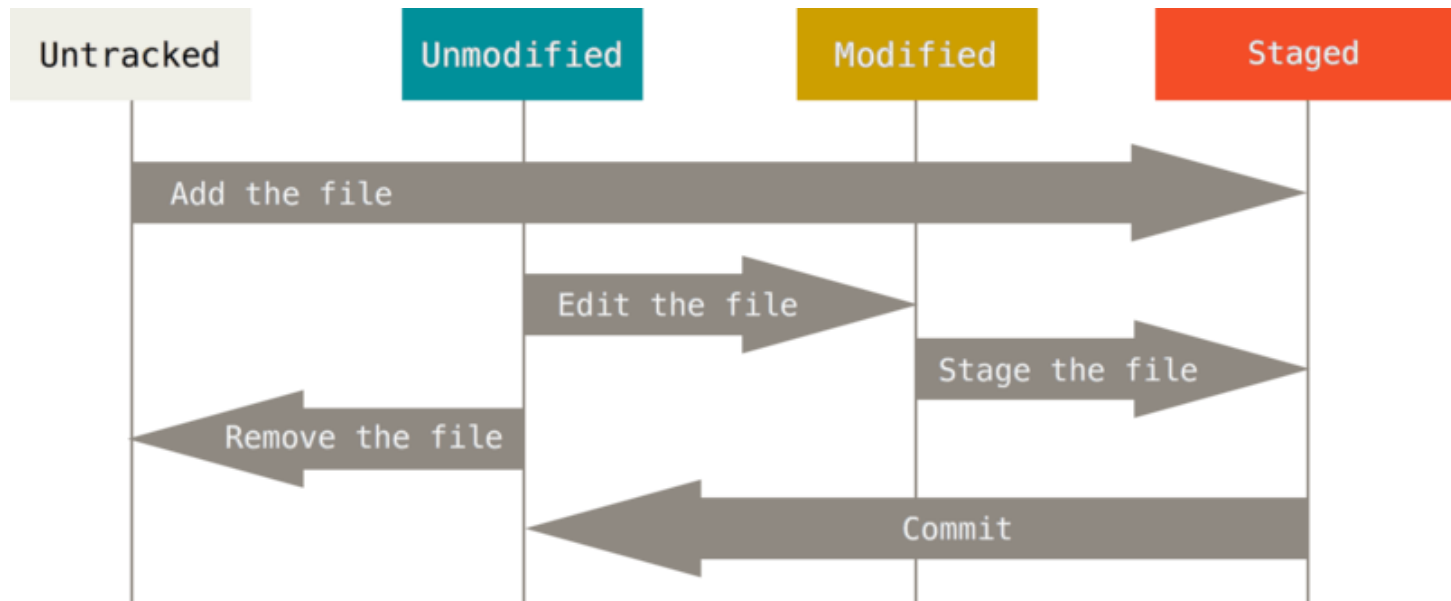
Git-Intern: 3 Hauptbereiche

- entferntes Repository ist in der Regel ein „*nacktes Repository*“ – ein Git-Repository ohne Arbeitsverzeichnis

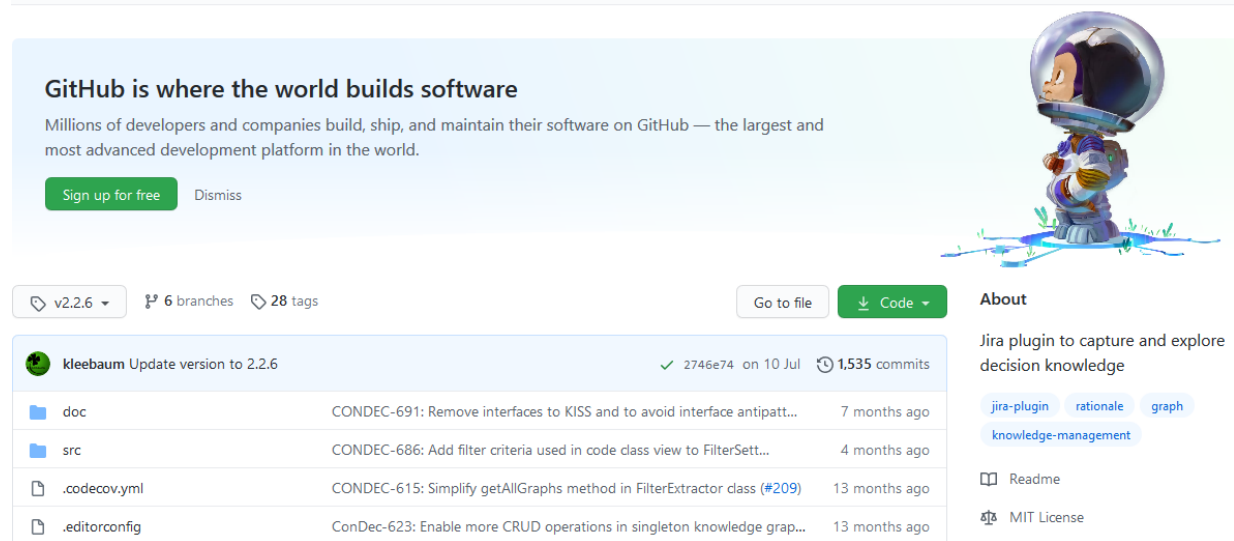


Git-Interns: Lebenszyklus einer Datei

- “tracked” Dateien sind entweder
 - modified (geändert) oder
 - unmodified (werden nicht in Commits eingebunden)
- Modified Dateien fließen nur dann in den nächsten Commit ein, wenn sie auch “staged” sind



- GitWeb einfache, webbasierte Git-Visualisierung (kein richtiger Git-Client, da keine Änderungen möglich)
- vollausgestattete webbasierte Git-Server:
 - GitLab
 - GitHub
 - Bitbucket



- Code-Reviews als Teil von Merge-/Pull-Requests
 - zeilenweise Diskussion der vorgeschlagenen Änderung
 - allgemeiner Diskussions-Thread

- Git-Tutorial in Confluence:
<https://confluence-se.ifi.uni-heidelberg.de/display/ISW2023/Versionsverwaltung+mit+Git>
- Offizielles Git-Tutorial:
<http://git-scm.com/docs/gittutorial>
- Buch “Pro Git”:
<https://git-scm.com/book>
- Atlassian Git-Anleitungen:
<https://www.atlassian.com/de/git/tutorials>
- Android Studio Git Tutorial:
<https://javapapers.com/android/android-studio-git-tutorial/>

Michael Anders

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 205
69120 Heidelberg, Germany
<https://se.ifi.uni-heidelberg.de>



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG
