

---

# Versionsmanagement

---



# Versionsmanagement

## Motivation

### Ausgangslage

- Softwareentwicklung ist Teamarbeit
  - Viel (indirekte) Kommunikation nötig
  - Entwicklungswissen muss dokumentiert werden
- Software besteht aus vielen Dokumenten
  - Lastenheft
  - Pflichtenheft
  - Analyse- und Designdokument
  - Programmcode
  - Dokumentation
  - Handbuch

# Versionsmanagement

## Motivation

### Konsequenz

- Verschiedene Personen greifen (gleichzeitig) auf Dokumente zu
- Oft bearbeiten verschiedene Personen gleichzeitig (unabhängig voneinander) das selbe Dokument

# Versionsmanagement

## Motivation

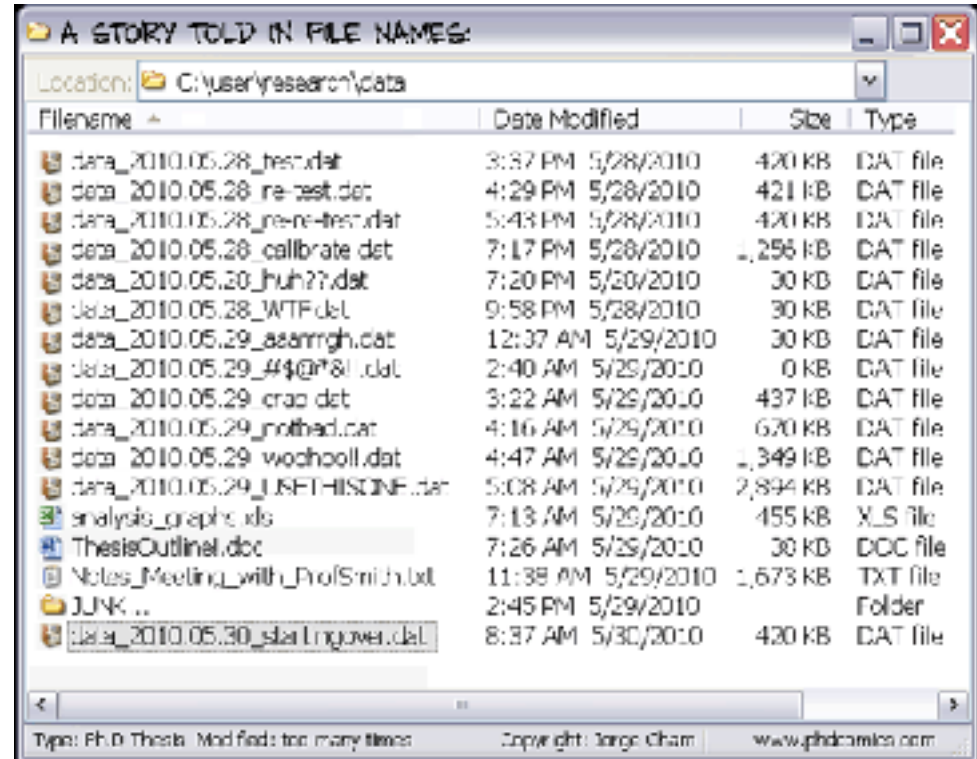
### Typische Probleme / Fragen

- Versionsmanagement
  - Wo ist die aktuelle Version?
  - Was ist die zuletzt lauffähige Version?
  - Wo ist Implementierungsversion vom 01. April 2016?
    - Und welche Dokumente beziehen sich auf diese Version?
  - Welche Version wurde dem Kunden „Bodden“ präsentiert?
- Änderungsmanagement
  - Was hat sich seit letzter Woche geändert?
  - Wer hat diese Änderung gemacht?
  - Warum wurde diese Änderung gemacht?

# Versionsmanagement

## Motivation

- Einfache Lösungen
  - Austausch der Dokumente via USB-Stick / Festplatte
  - Austausch der Dokumente via Mail
  - Netzwerkfestplatte
- Konventionen und Regeln werden im Team definiert



# Versionsmanagement

## Motivation

- **Einfache „Lösungen“ erzeugen neue Probleme**
  - Konventionen und Regeln werden nicht eingehalten
  - Koordination ist aufwendig und führt zu Verzögerungen
  - Varianten und Konfigurationen werden von Hand verwaltet
  - Versions- und Änderungsfragen nicht bzw. nur schwer beantwortbar
  - Geistige Kapazität wird mit „Kleinkram“ verschwendet
- **Fazit:**
  - Konventionen müssen technisch erzwungen werden!

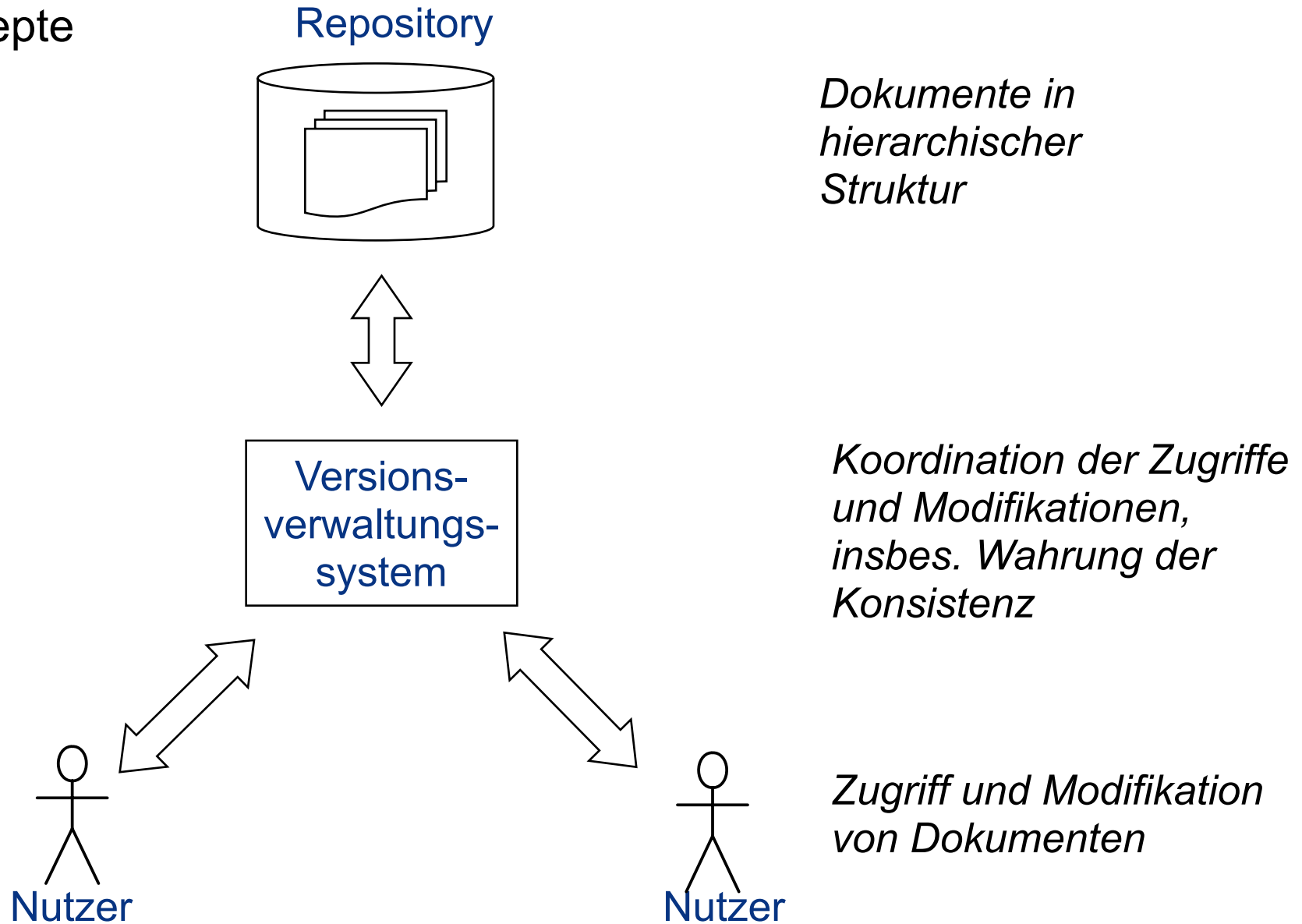
# Versionsmanagement

## Motivation

- **Sinnvolle Lösung**
  - Versions- und Konfigurationsmanagementsysteme
  - Lösen (bei vernünftiger Anwendung) alle genannten Probleme (fast) ohne Zusatzaufwand
  - Bieten sogar Zusatzleistungen (z.B. einfache Datensicherung)

# Versionsmanagement

## Konzepte





# Konsistenzmechanismen

- **Optimistische Mechanismen**

- System *erlaubt* gleichzeitiges Bearbeiten des Dokuments durch verschiedene Personen
- System erkennt und integriert die Änderungen (Merging)
- Evtl. funktioniert das nicht automatisch; dann muss der Konflikt manuell beseitigt werden

- **Pessimistische Mechanismen**

- System *verbietet* gleichzeitiges Bearbeiten des Dokuments durch verschiedene Personen (Sperrprotokolle)

- Beide Mechanismen haben Vor- und Nachteile

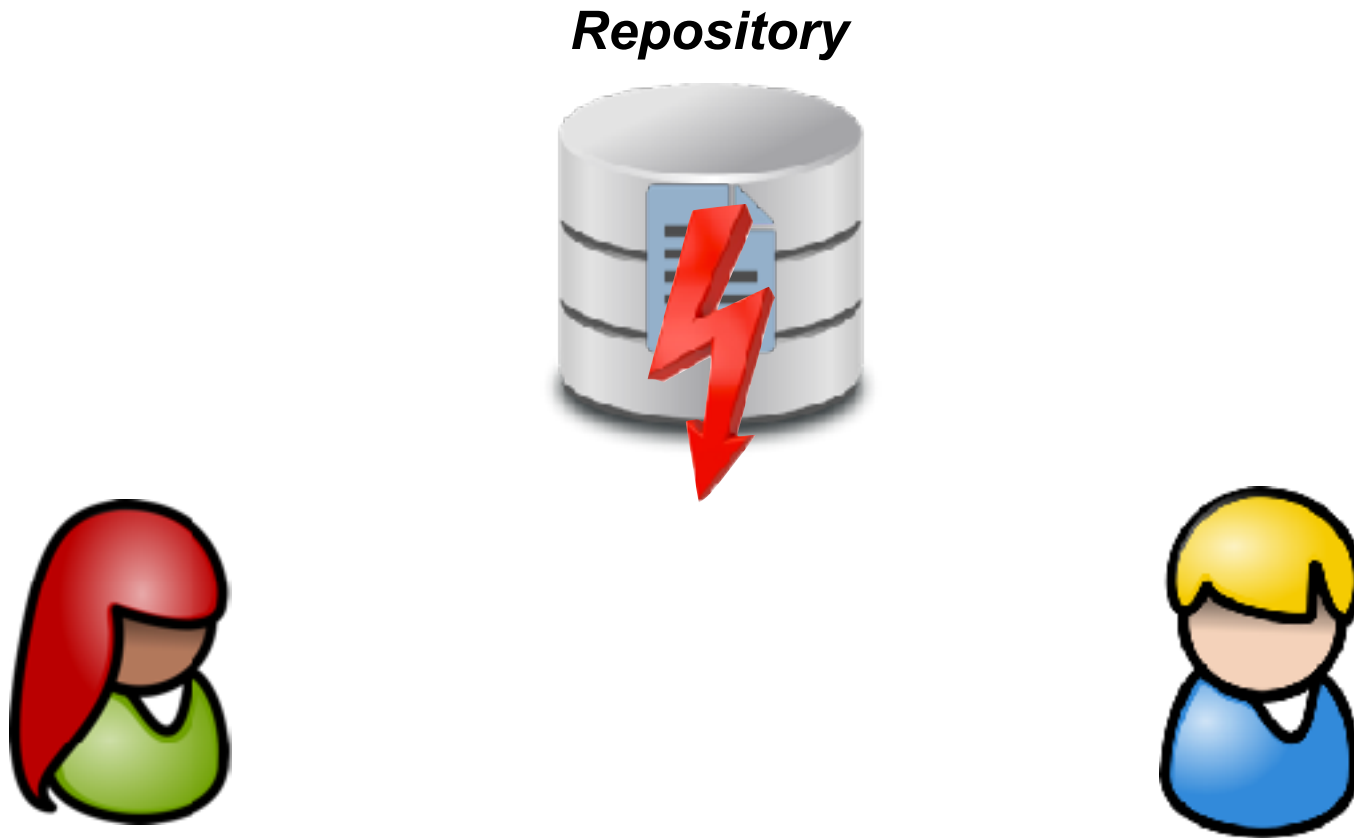
- Sperren serialisiert die Arbeit
- Mergen kann in seltenen Fällen komplex werden und zu Fehlern führen

# Zentrale Versionierung (z.B. CVS, SVN)

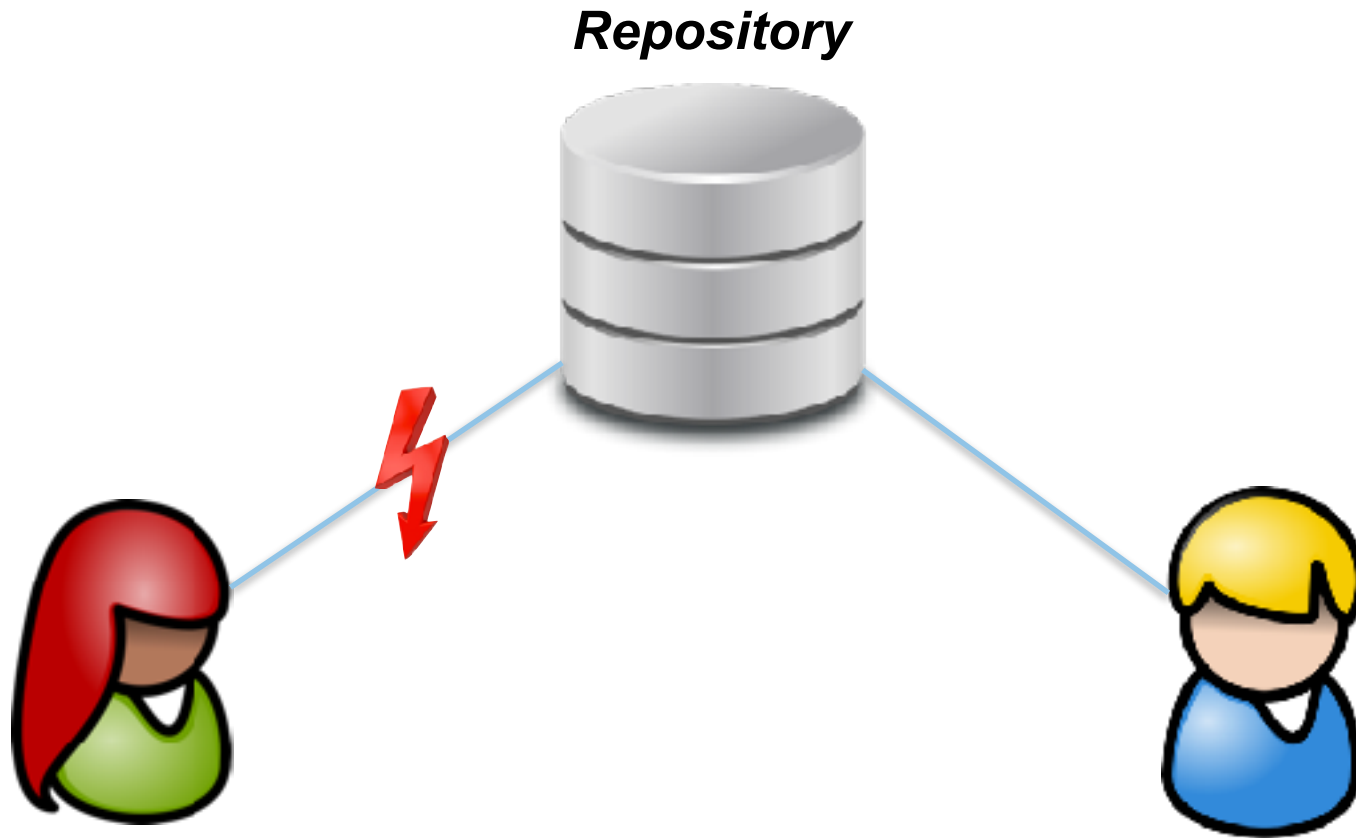
*Repository*



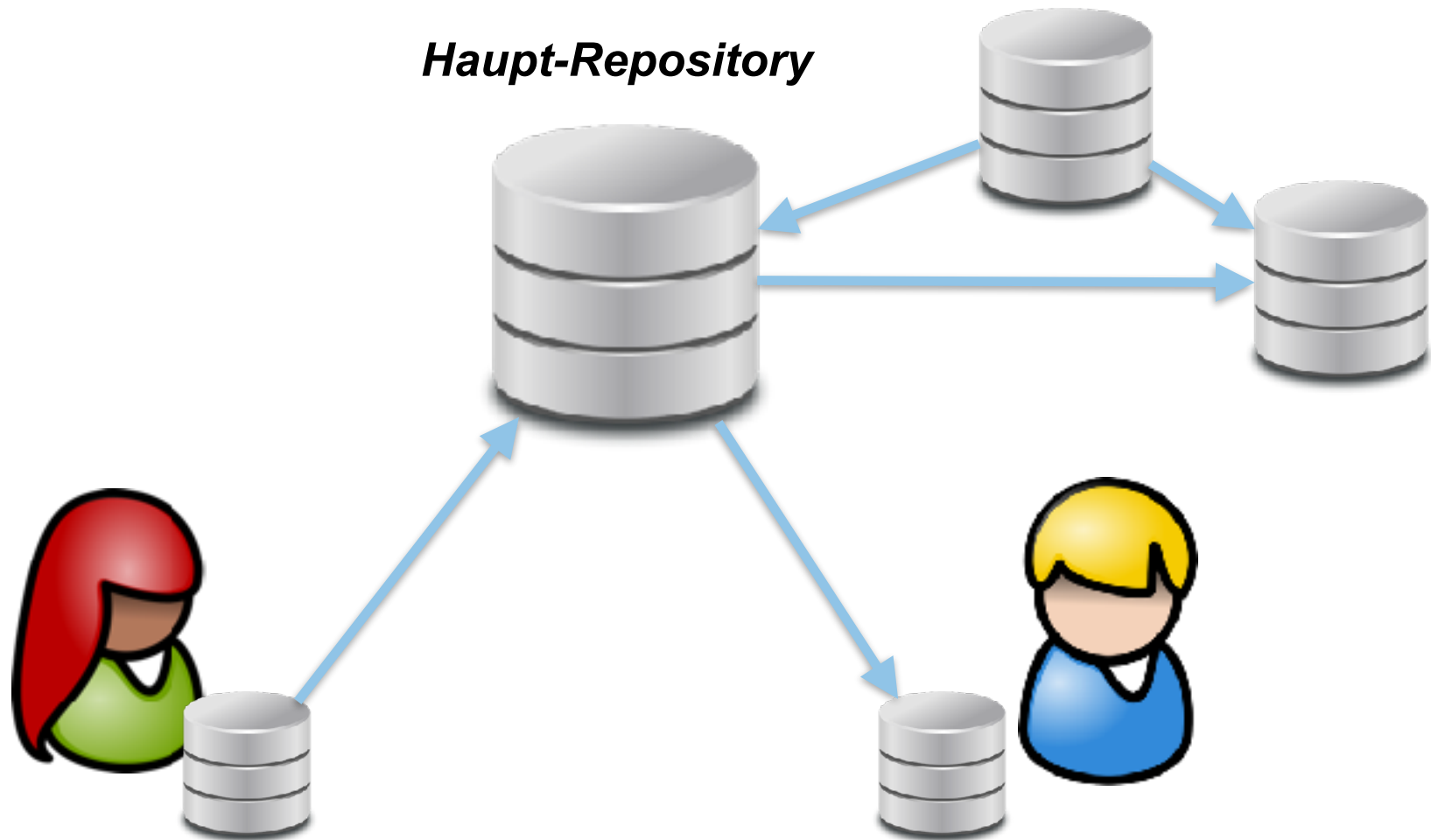
# Zentrale Versionierung - Problem: Konflikte



# Zentrale Versionierung - Problem: keine Offline-Nutzung möglich

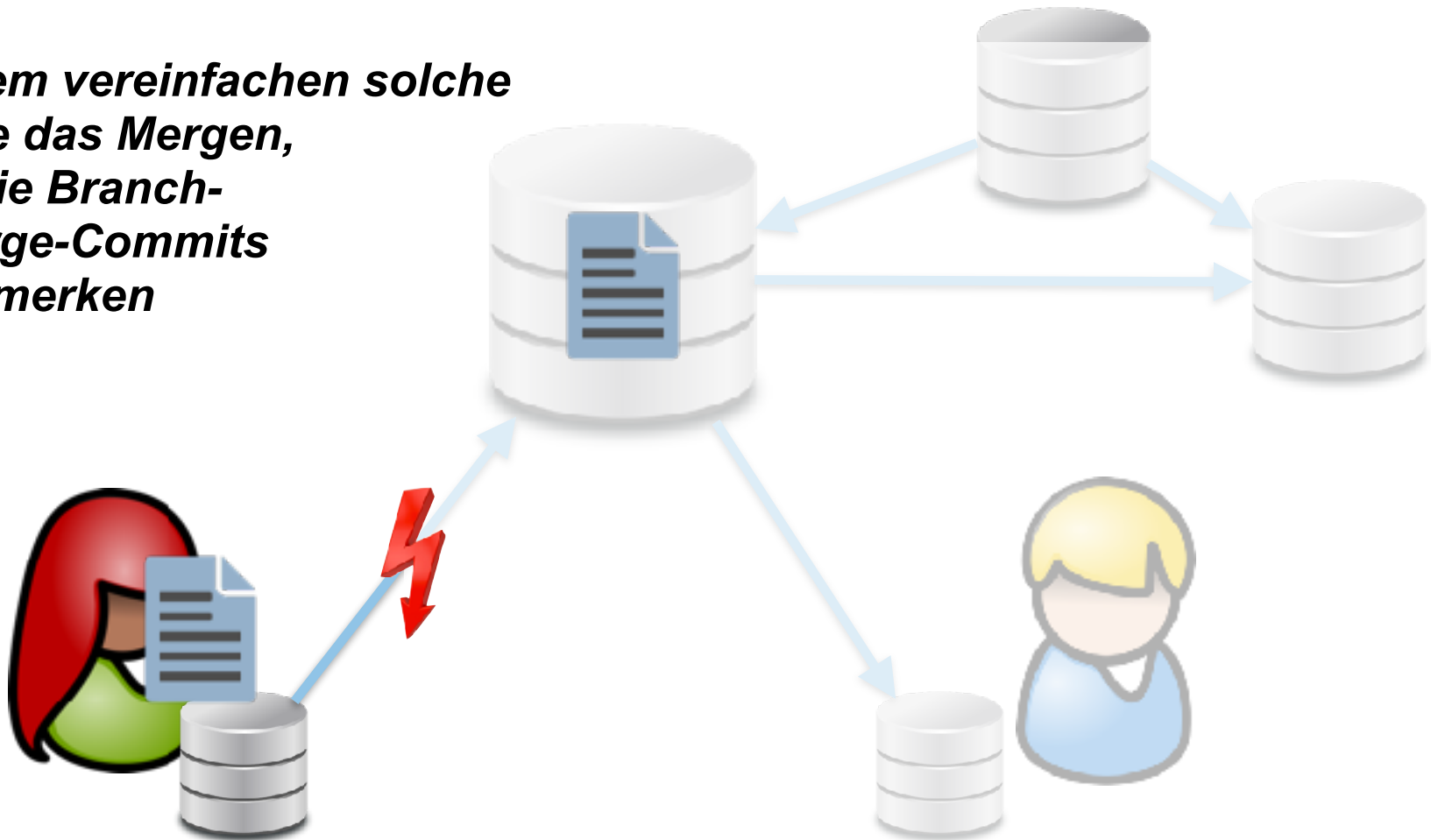


# Dezentrale Versionierung (z.B. Git, Mercurial, ...)



## Vorteil: Kann auch offline Dateien versionieren

*Außerdem vereinfachen solche Systeme das Mergen, indem sie Branch- und Merge-Commits explizit merken*



# In diesem Kurs verwenden wir Git

- Schritt 1: Git repository einrichten
  - Beispielsweise über Web-Frontend wie <https://git.cs.upb.de/>
- Schritt 2: Lokale Kopie des Remote-repositories “Klonen”:

```
git clone <repo-URL> [lokales Verzeichnis]
```

- Alternative: Repository direkt lokal anlegen:

```
git init
```

# Dateien neu Versionieren

- 1. Schritt: Dateien dem Repository hinzufügen

```
git add <Dateipfade>
```

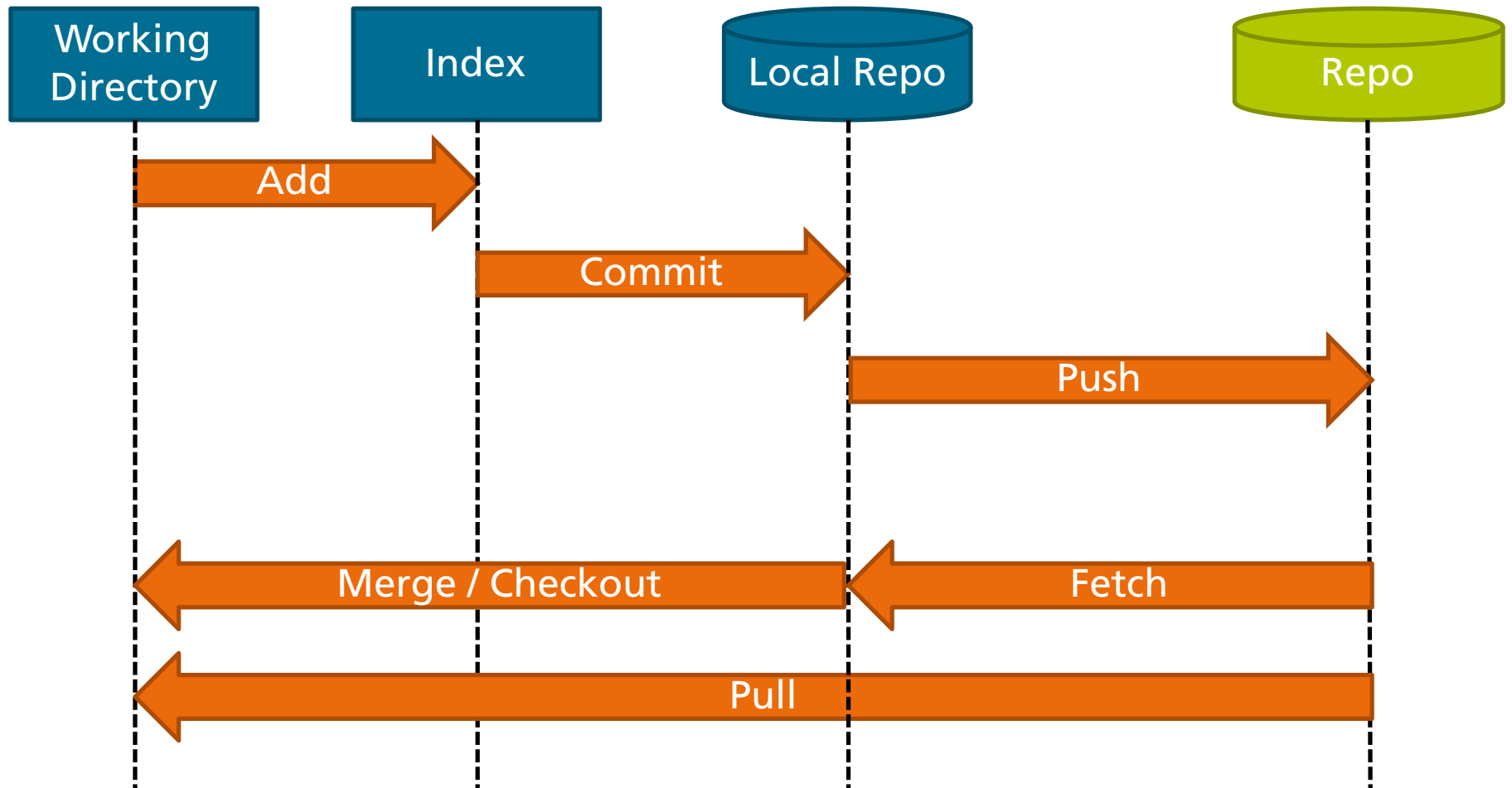
- Dateien landen dann in der sogenannten “Staging Area”
- Die “Staging Area” (oder *Index*) hält alle Änderungen, Hinzufügungen und Löschungen von Dateien, die Teil des nächsten Commits werden sollen

- 2. Schritt: gestagte Änderungen committen

```
git ci
```

- Dies fügt die Änderungen dem **lokalen** Repository zu





# Git verwaltet Versionen des kompletten Repositories mittels Commits, Branches und Tags



*git rm*

*git add*

*git add*

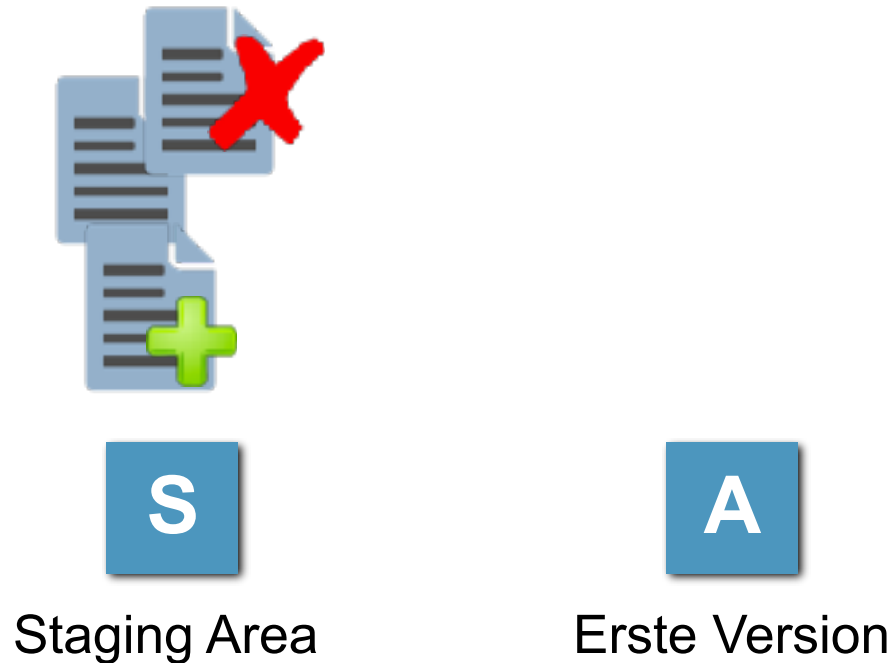
*auch Änderungen muss man stagen!*



Staging Area

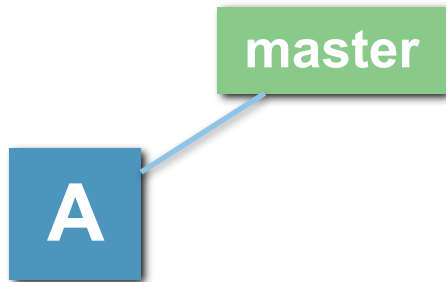
# Git verwaltet Versionen des kompletten Repositories mittels Commits, Branches und Tags

*git commit*



# Git verwaltet Versionen des kompletten Repositories mittels Commits, Branches und Tags

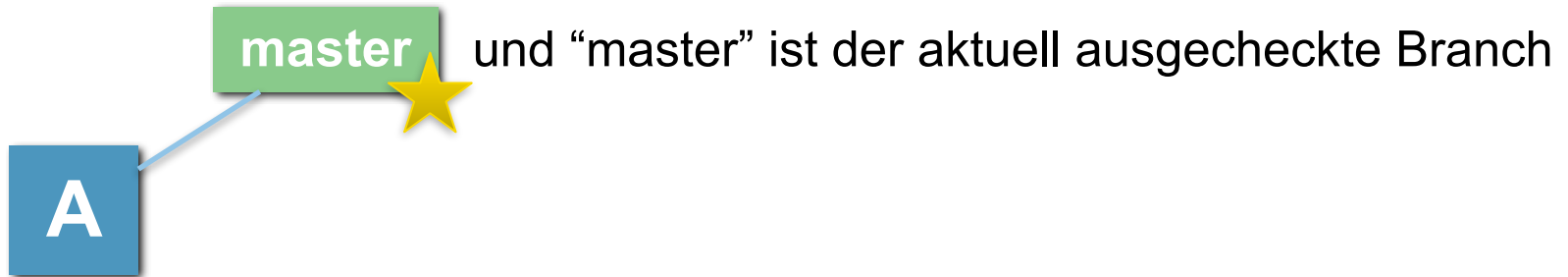
## *Initiales Setup:*



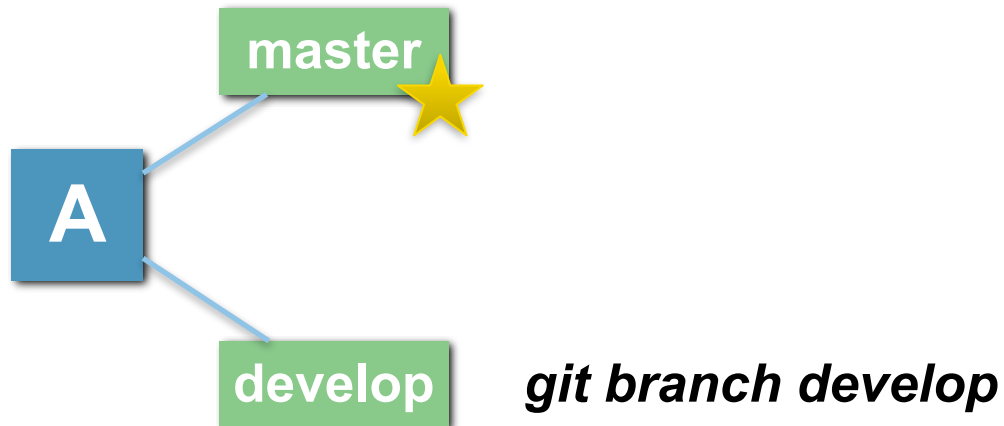
Branch “master”, ist aktuell auf dieser Version

# Git verwaltet Versionen des kompletten Repositories mittels Commits, Branches und Tags

## *Initiales Setup:*



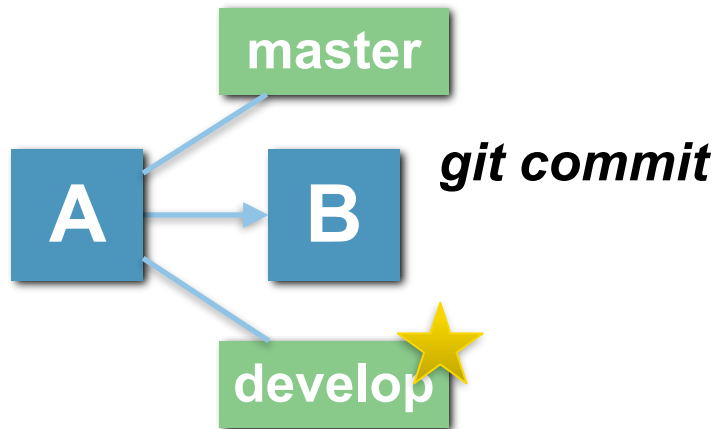
# Branching



# Branch auschecken



## Neuer Commit “B”

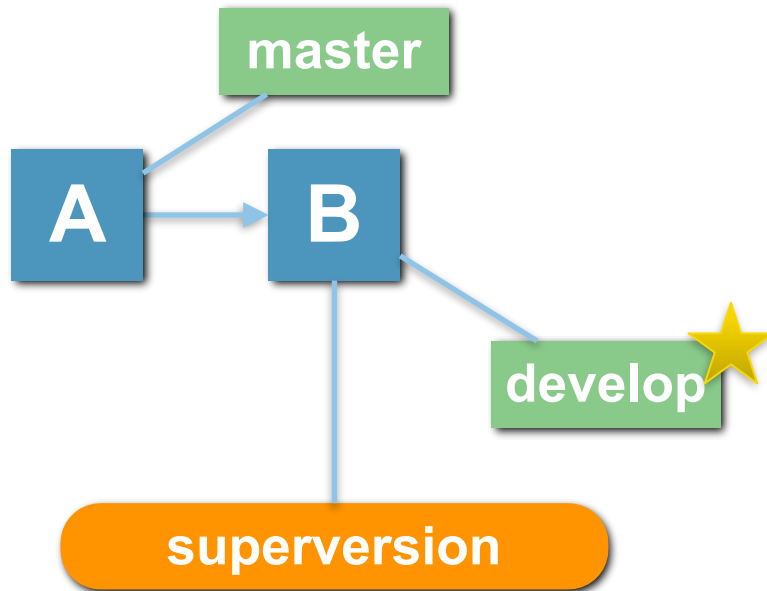




## Neuer Commit “B”

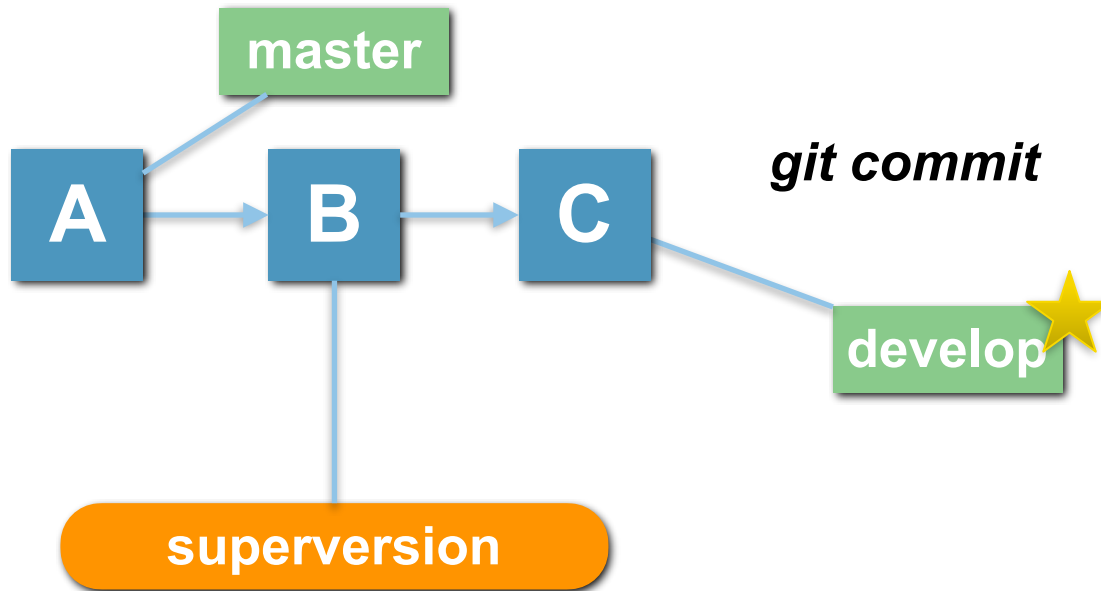


# Tagging



*git tag supervision*

## Neuer Commit "C"



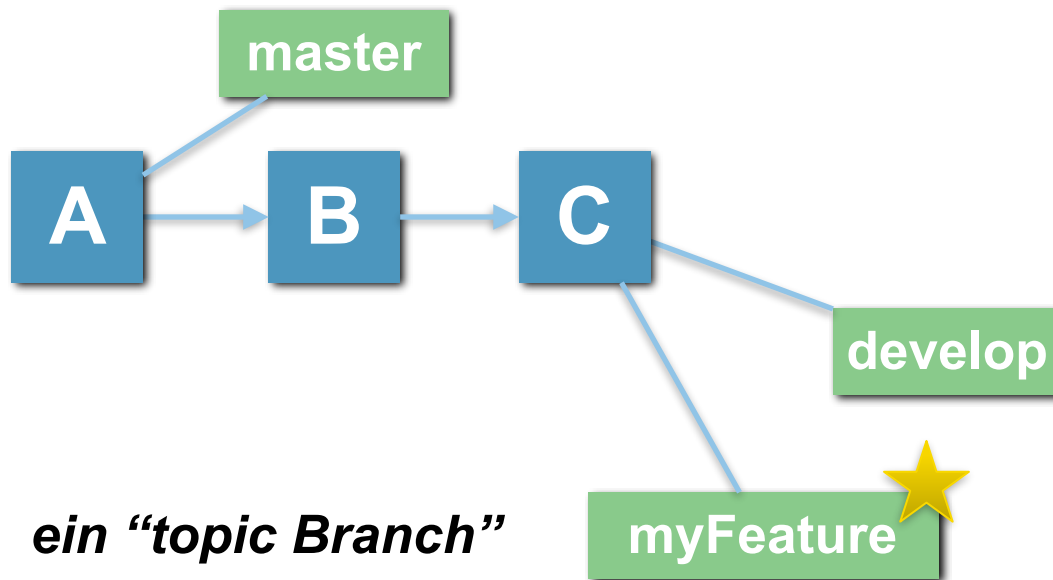
# Arbeiten mit Branches

***git checkout -b myFeature***

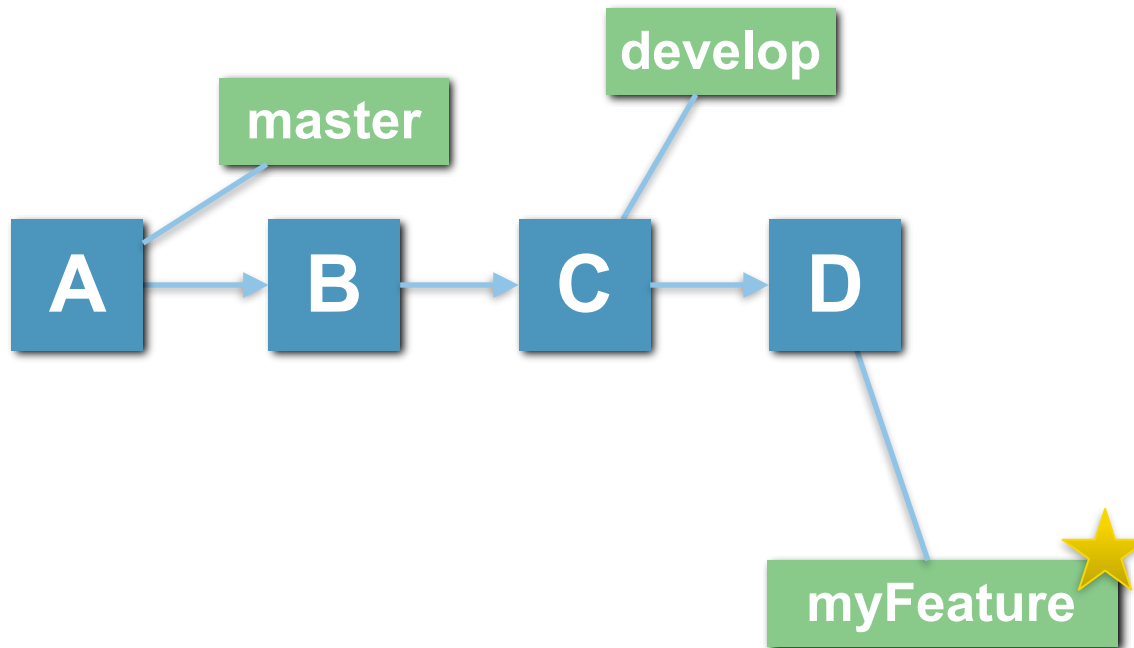
*kurz für:*

***git branch myFeature***

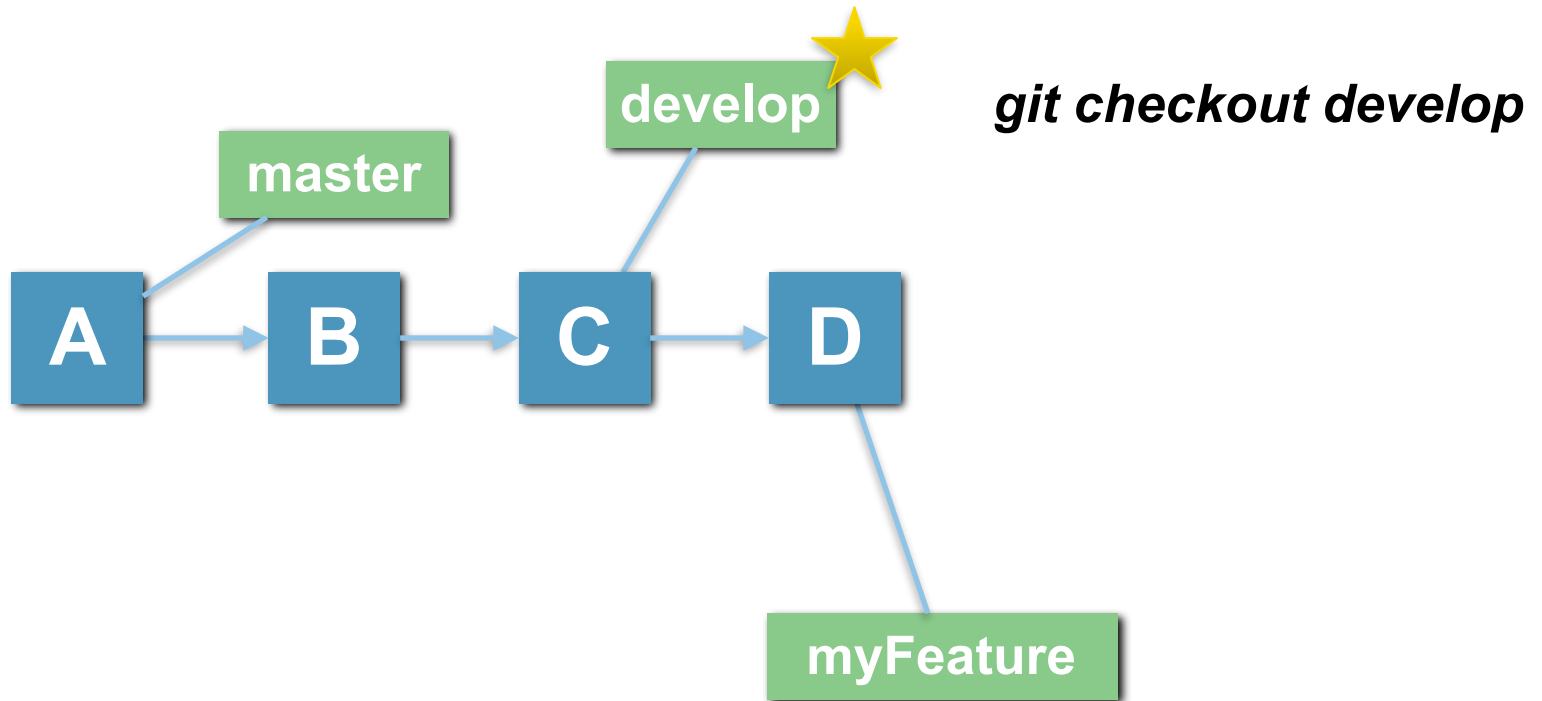
***git checkout myFeature***



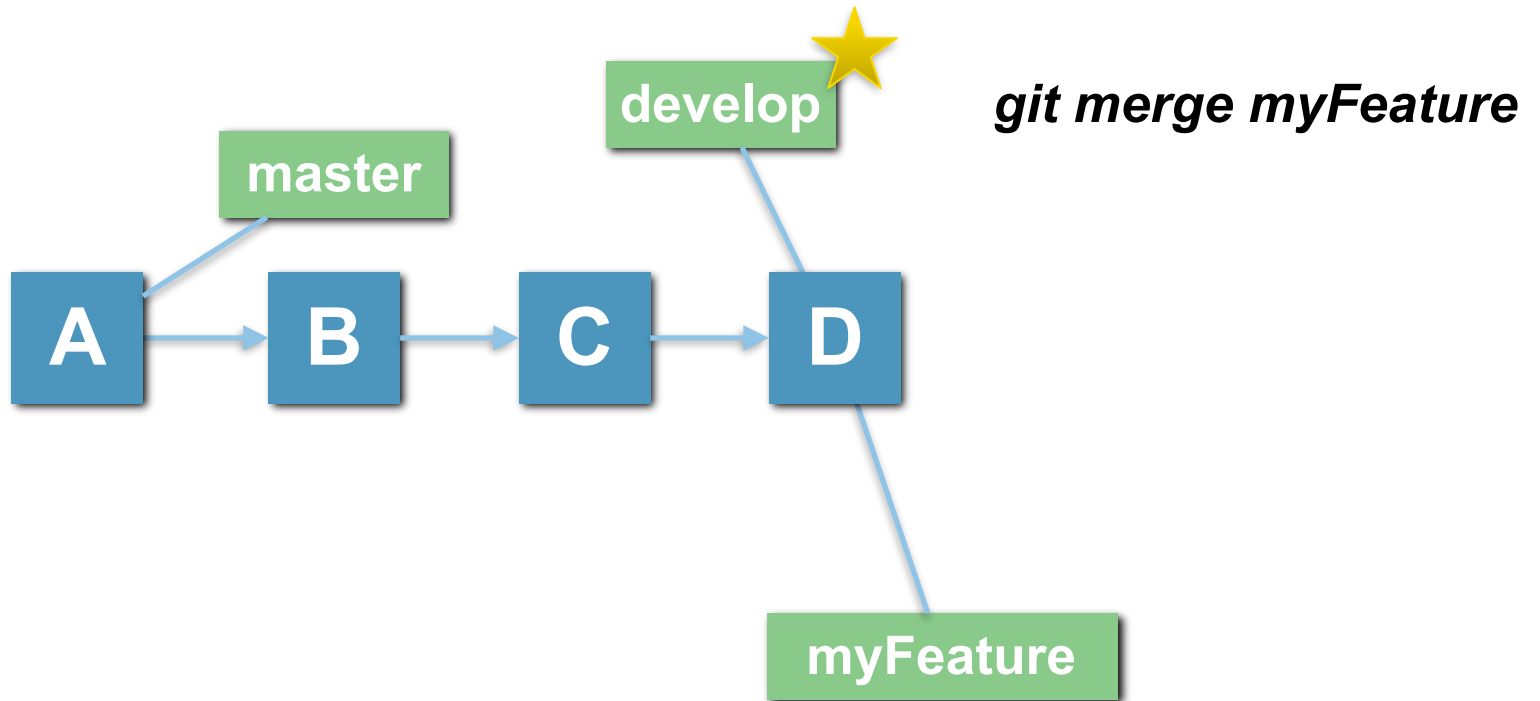
## Arbeiten mit Branches - Neuer Commit "D"



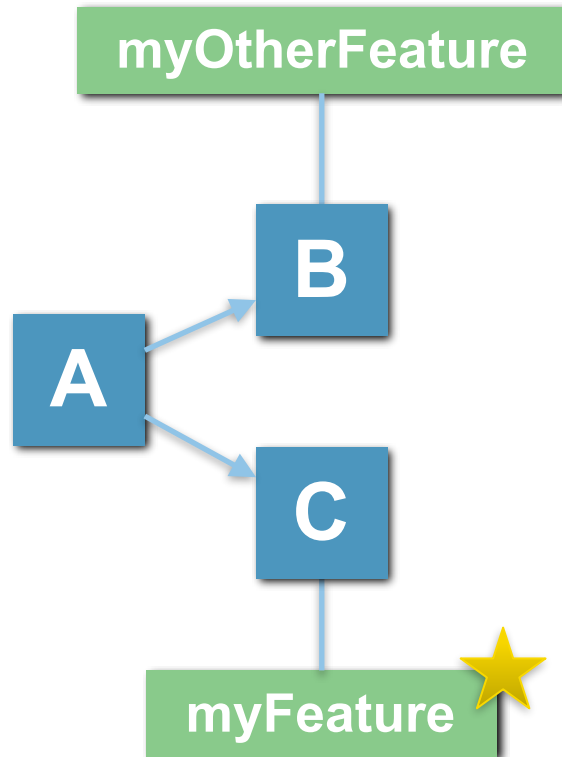
# Arbeiten mit Branches



# Fast-Forward Merge



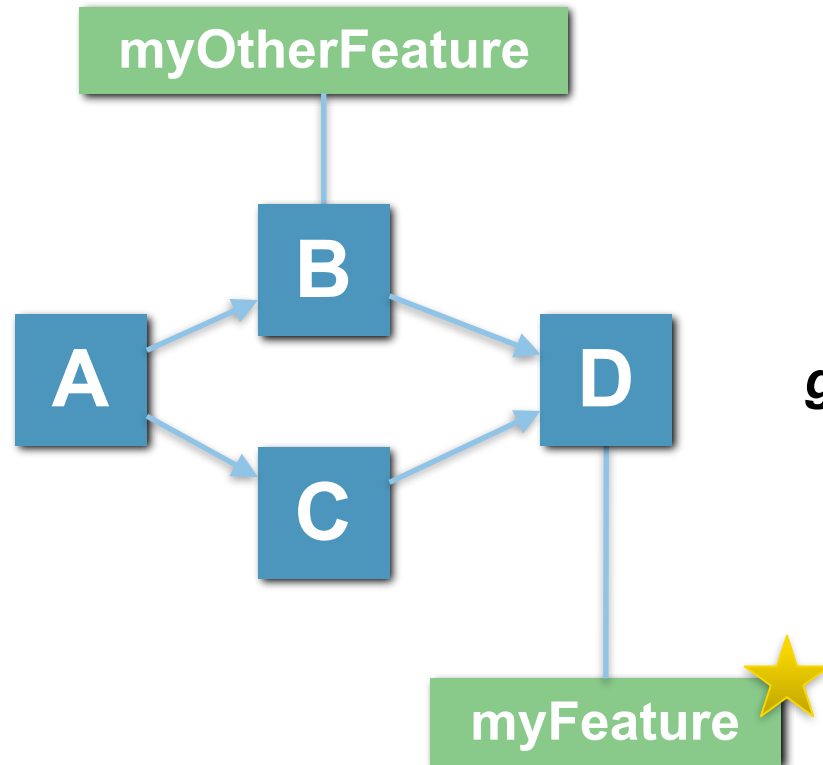
# Komplexerer Merge



*Situation vor dem Merge*

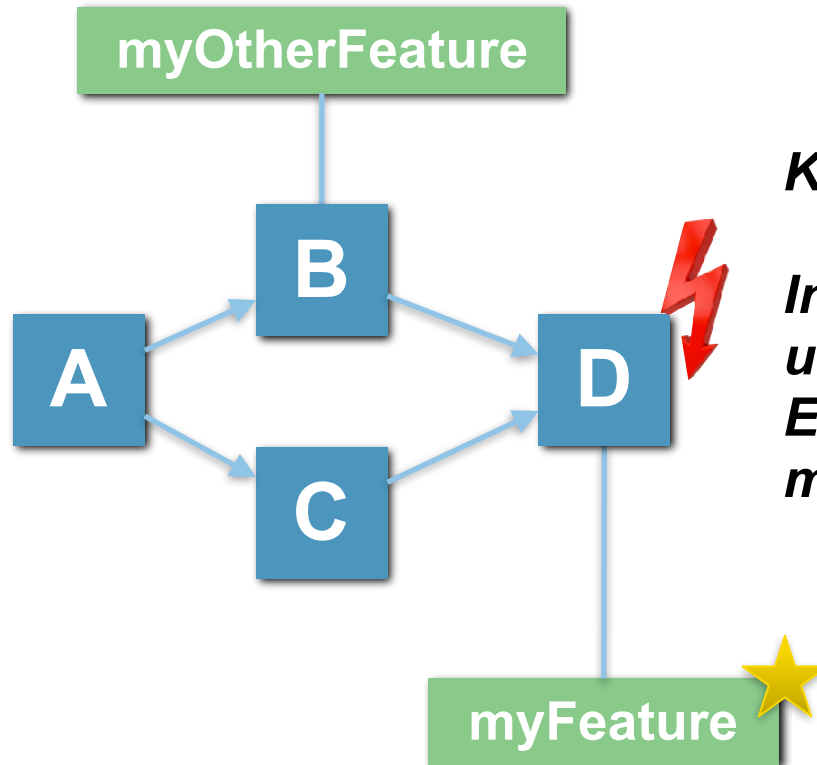


# Komplexerer Merge



*git merge myOtherFeature*

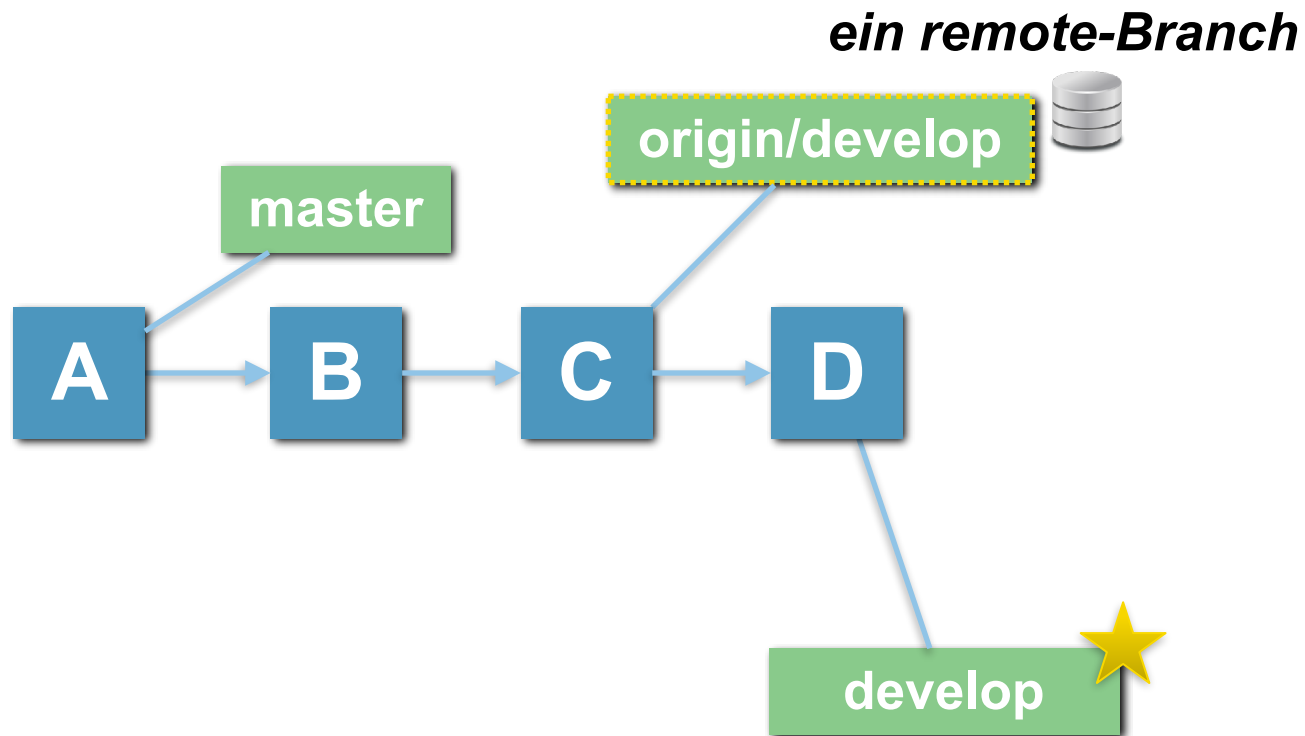
# Komplexerer Merge



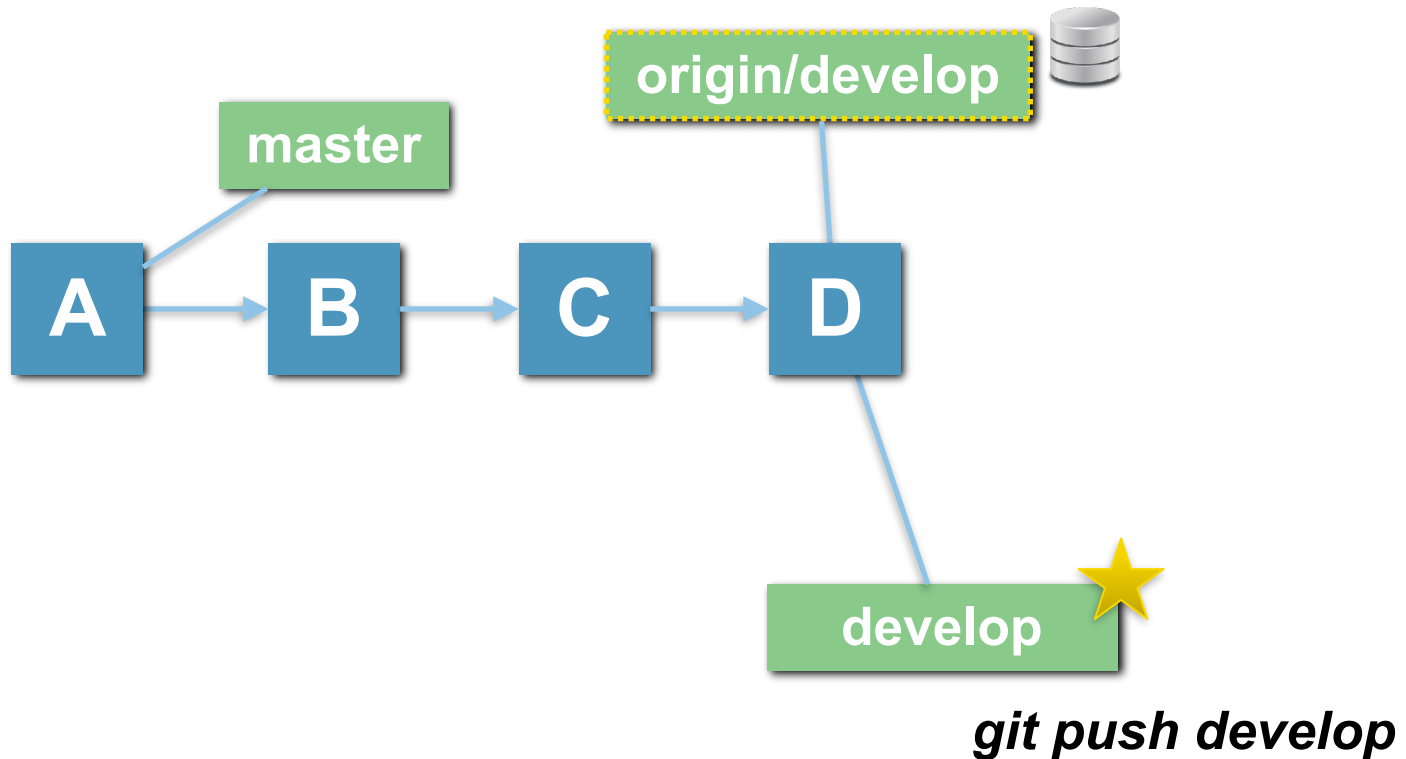
***Kann zu Konflikten führen!***

***In dem Fall wird “D” nur gestaged und nicht committed.  
Entwickler muss dann händisch mergen und committen.***

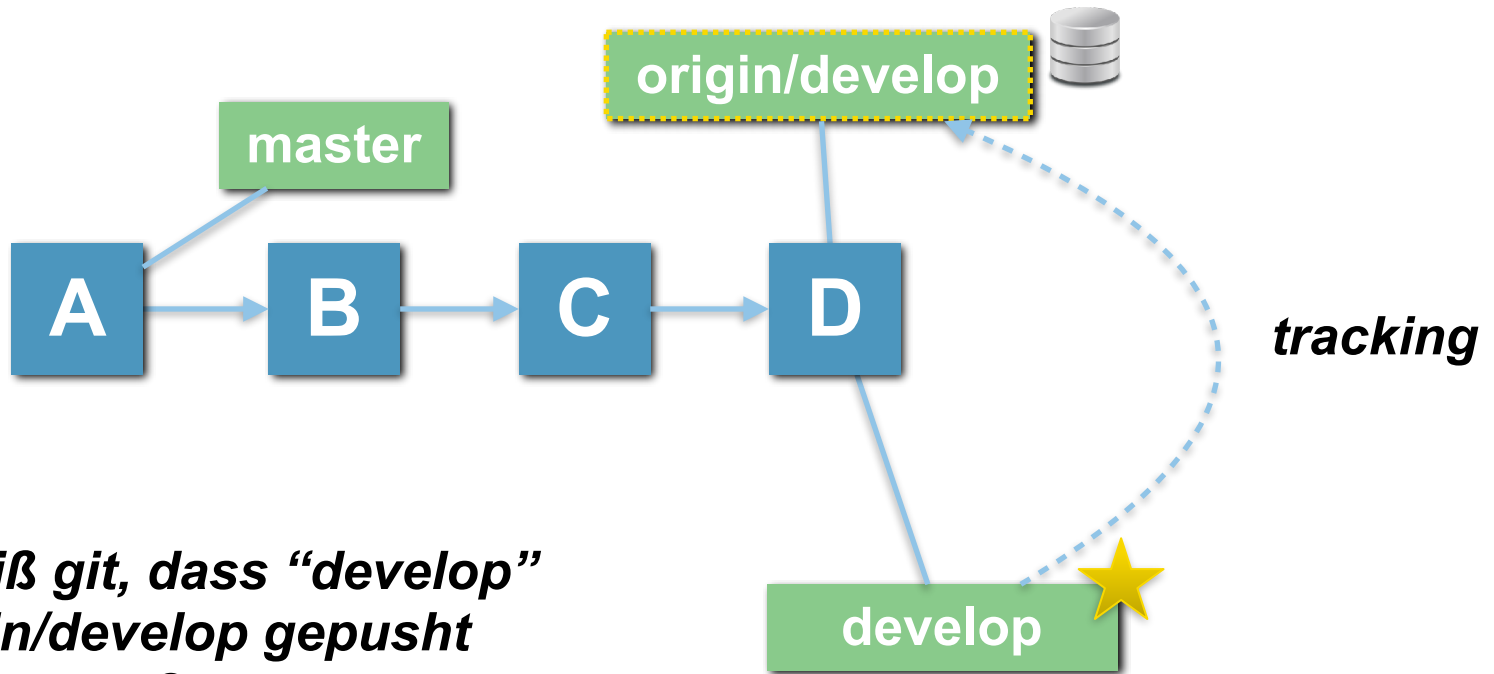
# Änderungen an Repository senden



# Änderungen an Repository senden



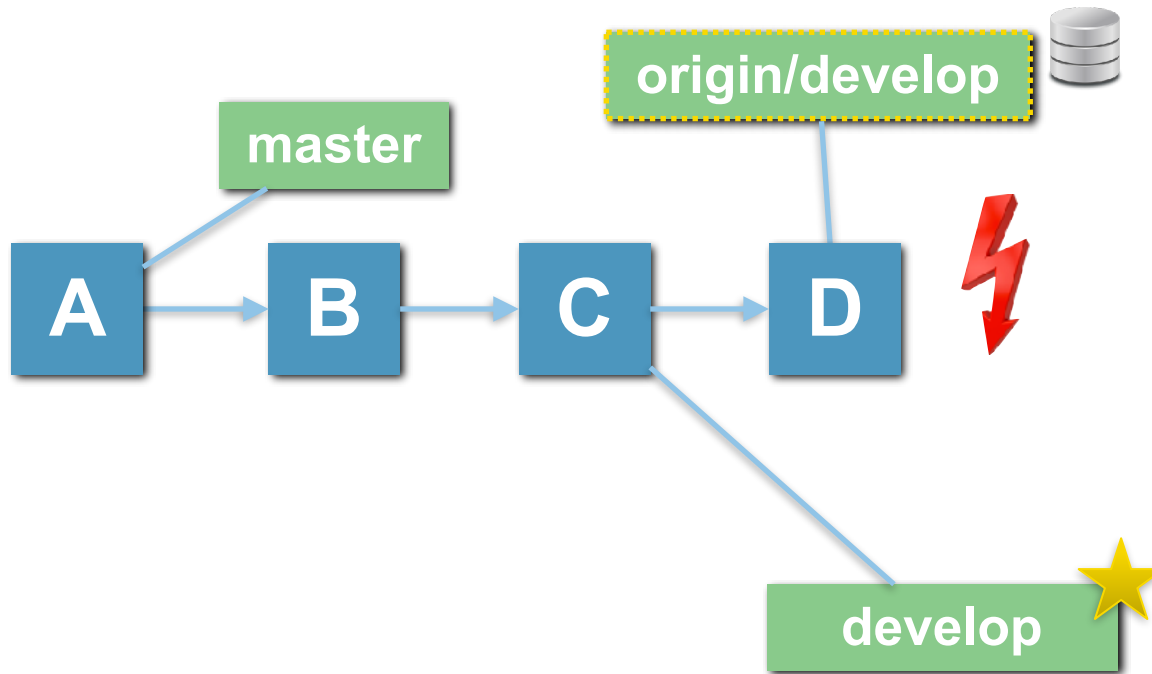
# Änderungen an Repository senden



***Wie weiß git, dass “develop” zu origin/develop gepusht werden muss?***

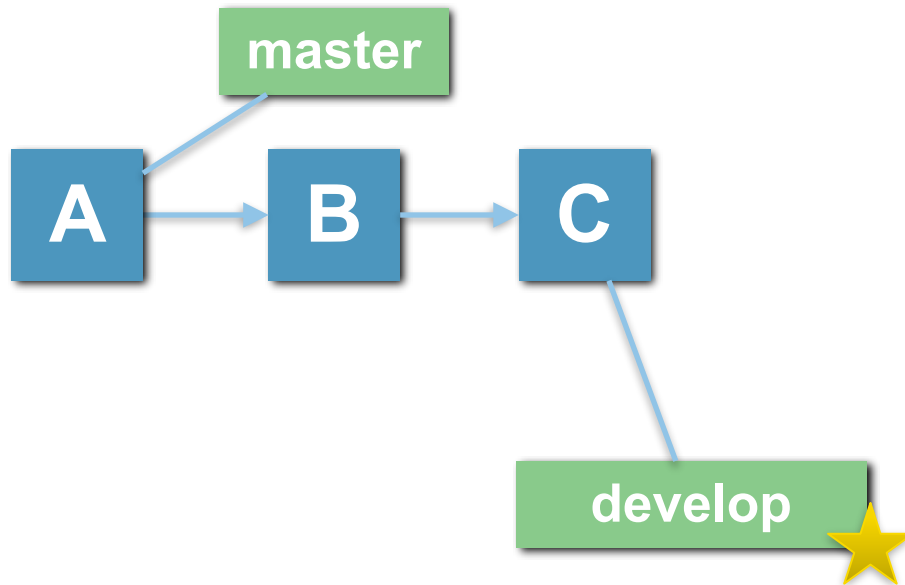
***“develop” is ein “tracking branch” für origin/develop***

# Kann nur konfliktfrei pushen!

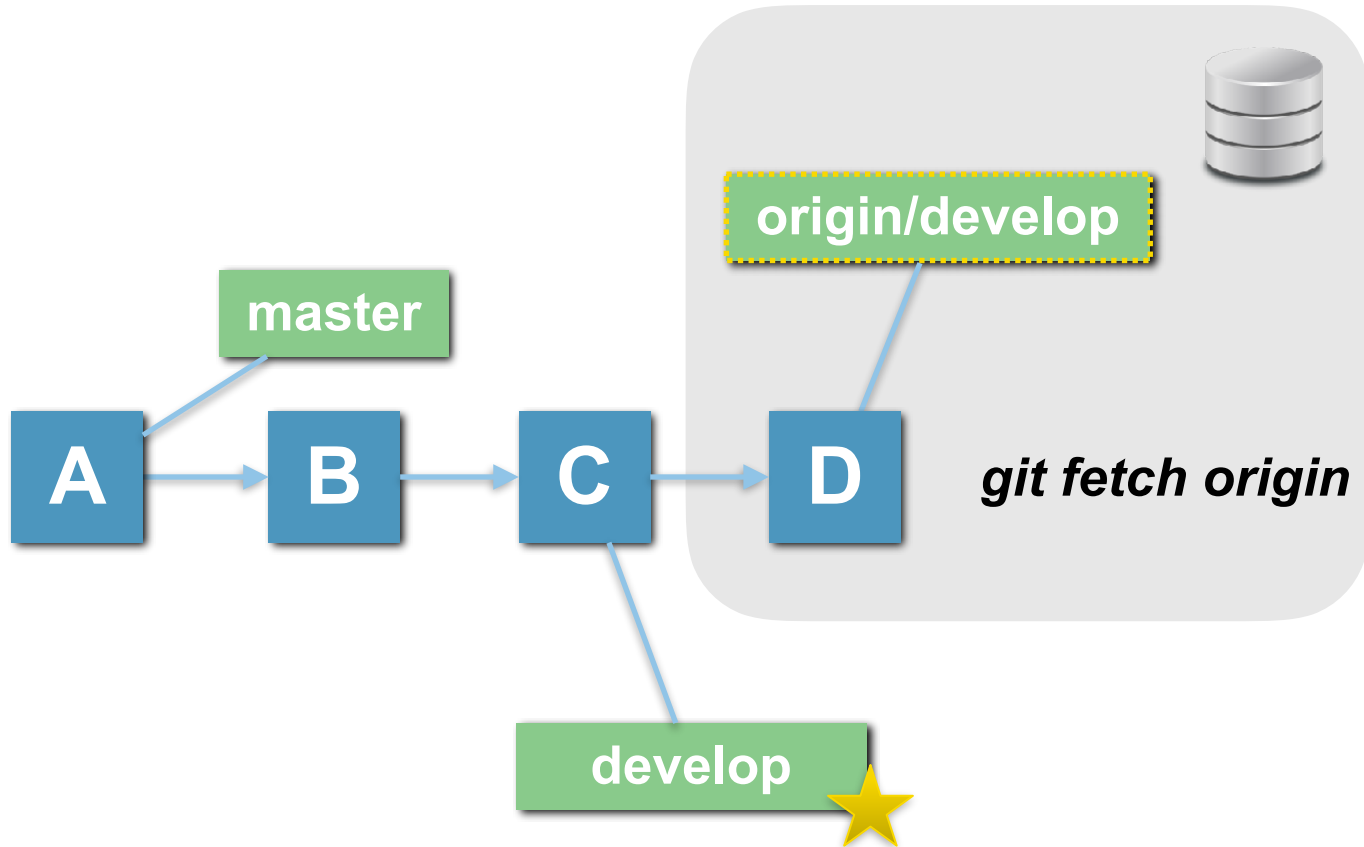


***“git push” funktioniert hier nicht***

# Änderungen empfangen

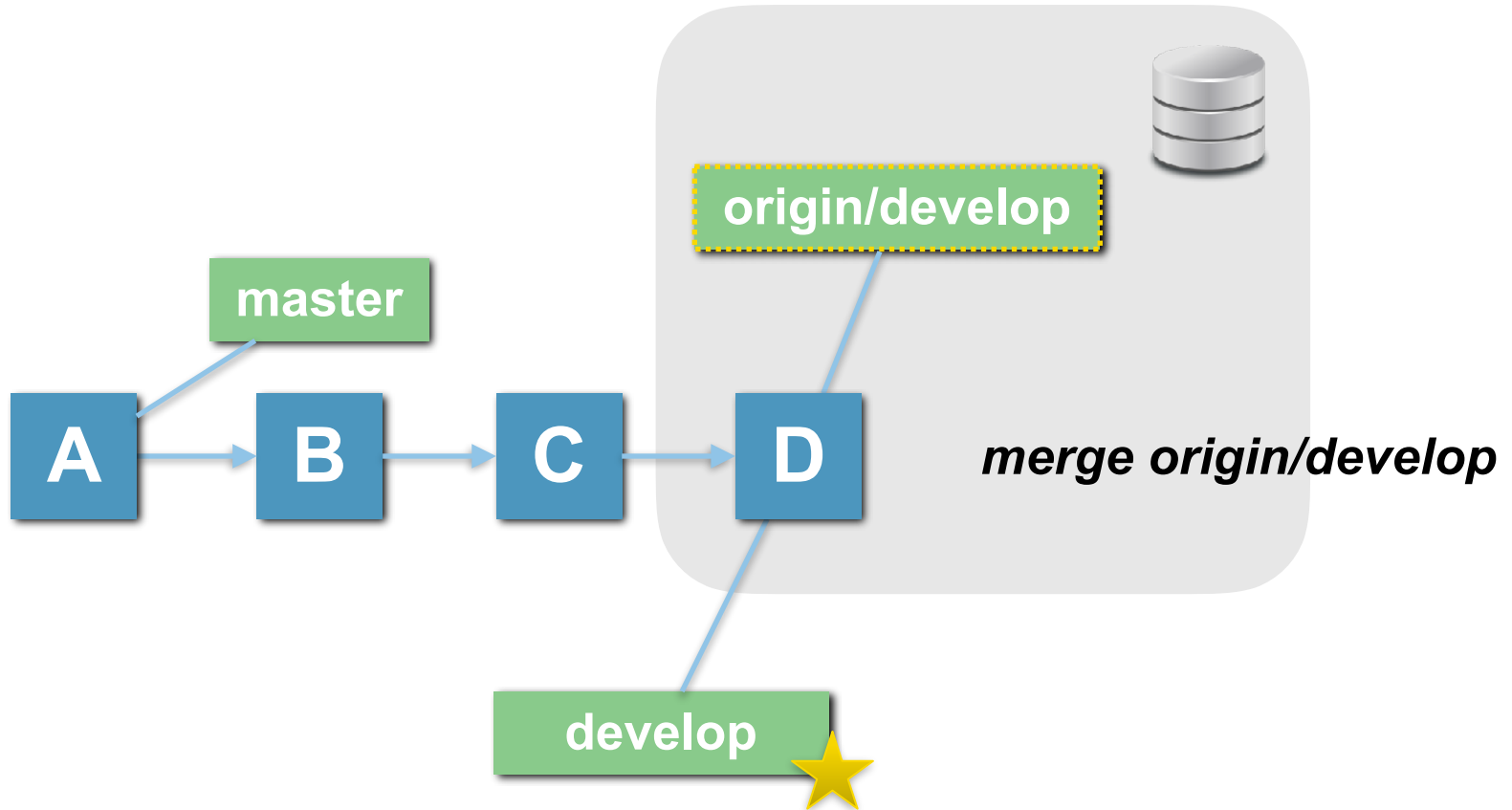


# Änderungen empfangen





# Änderungen empfangen



**Kurzversion: *git pull***

**entspricht: *git fetch origin; git merge origin/develop***

# Weitere hilfreiche Kommandos

- Aktuelle Änderungen zwischenspeichern und Working Copy resettet:

```
git stash
```

Hilfreich z.B. wenn man vergessen hat, Änderungen von remote zu “pullen”. Ein “pull” könnte lokale Änderungen überschreiben, mit “git stash” werden diese Änderungen aber zunächst sicher beiseite gelegt.

- Änderungen vom “Stash” in Working Copy zurückspielen:

```
git stash pop
```

- Änderungen in der Working Copy zurücksetzen (z.B. nach einem “Fehlversuch”)

```
git reset --hard
```

Setzt **alle** Änderungen in der Working Copy auf den letzten Commit zurück. Andere Varianten des Befehls (z.B. *soft*) sind ebenfalls verfügbar.

- Username und Emailadresse als Metadaten für Commits setzen:

```
git config user.name <name>  
git config user.email <e-mail>
```

# SSH-Schlüssel

- Github, Bitbucket, andere Hosters und auch GitLab ([git.cs.upb.de](https://git.cs.upb.de)) unterstützen den Zugriff per HTTPS und SSH
- SSH bietet den Vorteil, dass man sich ohne die Eingabe eines Passworts mit dem Server verbinden kann
  - die Authentifizierung geschieht in diesem Fall mittels eines geheimen Schlüssels, der auf dem Client-PC gespeichert wird
  - Nachteil: stiehlt ein Trojaner den Schlüssel, hat der Angreifer Zugriff auf das Repository
- Schlüsselpaar (private und public key) generieren (Länge 4096 bytes):

```
ssh-keygen -b 4096
```

- Öffentlicher Schlüssel liegt dann unter:

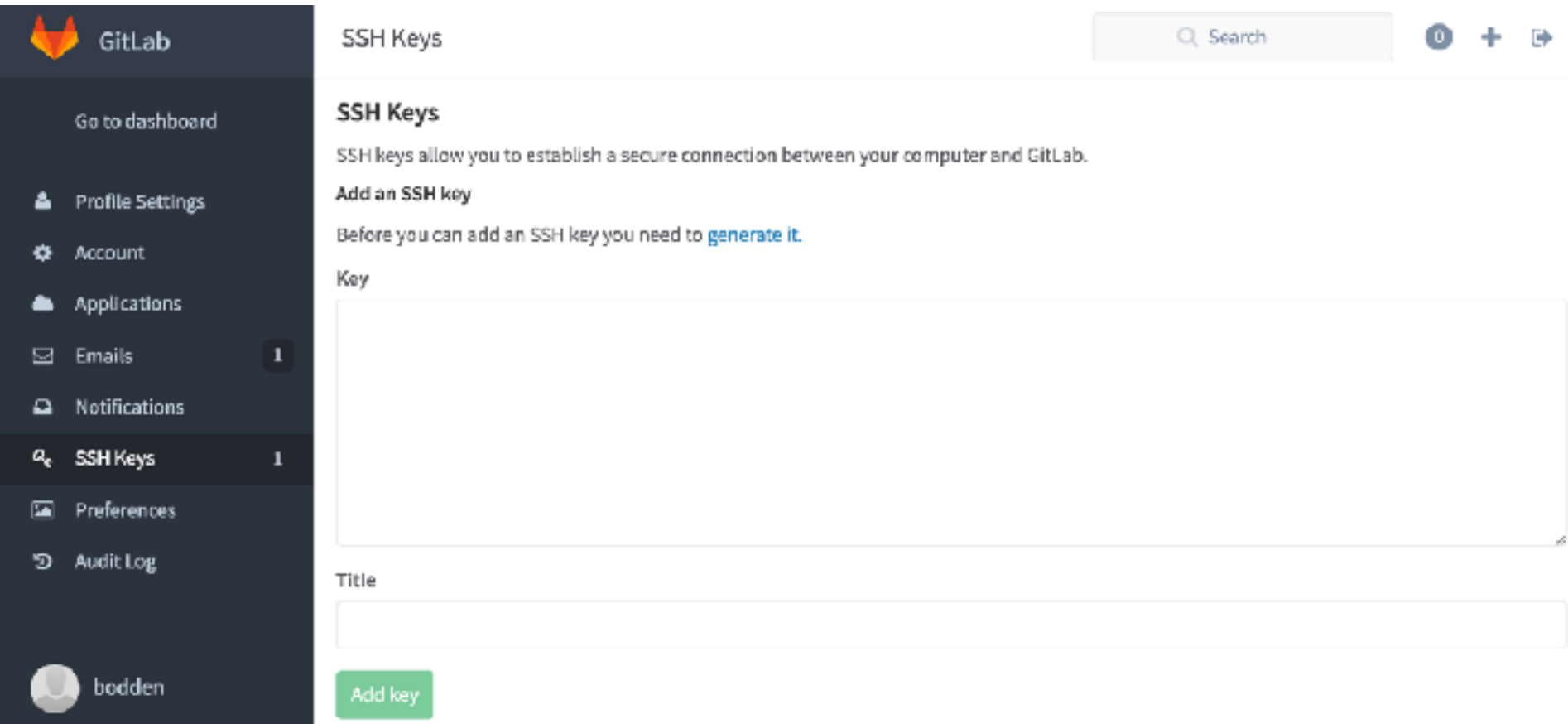
```
~/.ssh/id_rsa.pub
```

- Privater Schlüssel (geheim halten!) liegt unter:

```
~/.ssh/id_rsa
```

- Den öffentlichen Schlüssel kann man mit einem Texteditor einsehen und dann beim Hosting-Dienst in das entsprechende Formular pasten.

# Beispiel: SSH-Schlüssel in GitLab eintragen



The screenshot shows the GitLab user interface. On the left is a dark sidebar with the GitLab logo and a menu: 'Go to dashboard', 'Profile Settings', 'Account', 'Applications', 'Emails' (with a '1' badge), 'Notifications', 'SSH Keys' (with a '1' badge), 'Preferences', and 'Audit Log'. At the bottom of the sidebar is a user profile for 'bodden'. The main content area is titled 'SSH Keys' and includes a search bar, a notification icon, and a plus icon. Below the title, it says 'SSH keys allow you to establish a secure connection between your computer and GitLab.' and 'Add an SSH key'. A note states: 'Before you can add an SSH key you need to [generate it](#).' There is a large text input field labeled 'Key' and a smaller one labeled 'Title'. A green 'Add key' button is at the bottom left of the main area.

***Tipp: Man kann einen Schlüssel für mehrere Dienste verwenden, allerdings mit dem üblichen Nachteil, falls der Schlüssel kompromittiert wird.***

# Git-Flow Konventionen

- Git-Flow ist eine Konvention zur Nutzung von Branches in einer sinnvollen Art und Weise
- Erstmals dokumentiert durch Vincent Driessen  
<http://nvie.com/posts/a-successful-git-branching-model/>
- Mindestens fünf Arten von Branches:
  - **master:** enthält stets die zuletzt releaste Version
  - **develop:** enthält aktuelle Entwicklungsversion
  - **feature/topic branches:** zur Entwicklung individueller Features
  - **hotfix branches:** zur Implementierung dringender Bugfixes
  - **release branches:** zum Vorbereiten eines Releases

# Feature branches oder Topic branches

- Herkunft:
  - develop
- Wird gemerged in:
  - develop
- Namenskonvention:
  - irgend etwas außer *master*, *develop*, *release-\**, *hotfix-\**

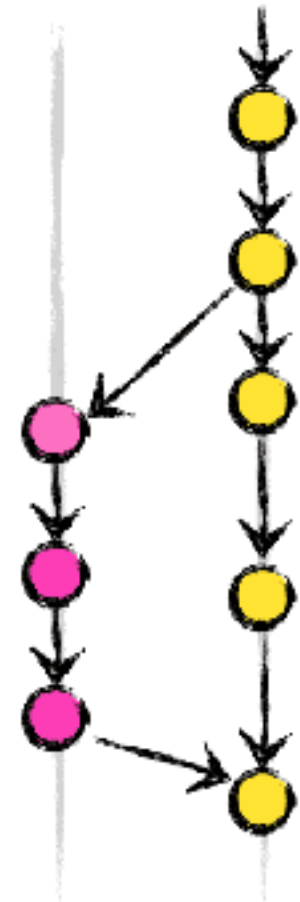
- Topic-branch erzeugen mittels:

```
git checkout develop  
git checkout -b myTopic
```

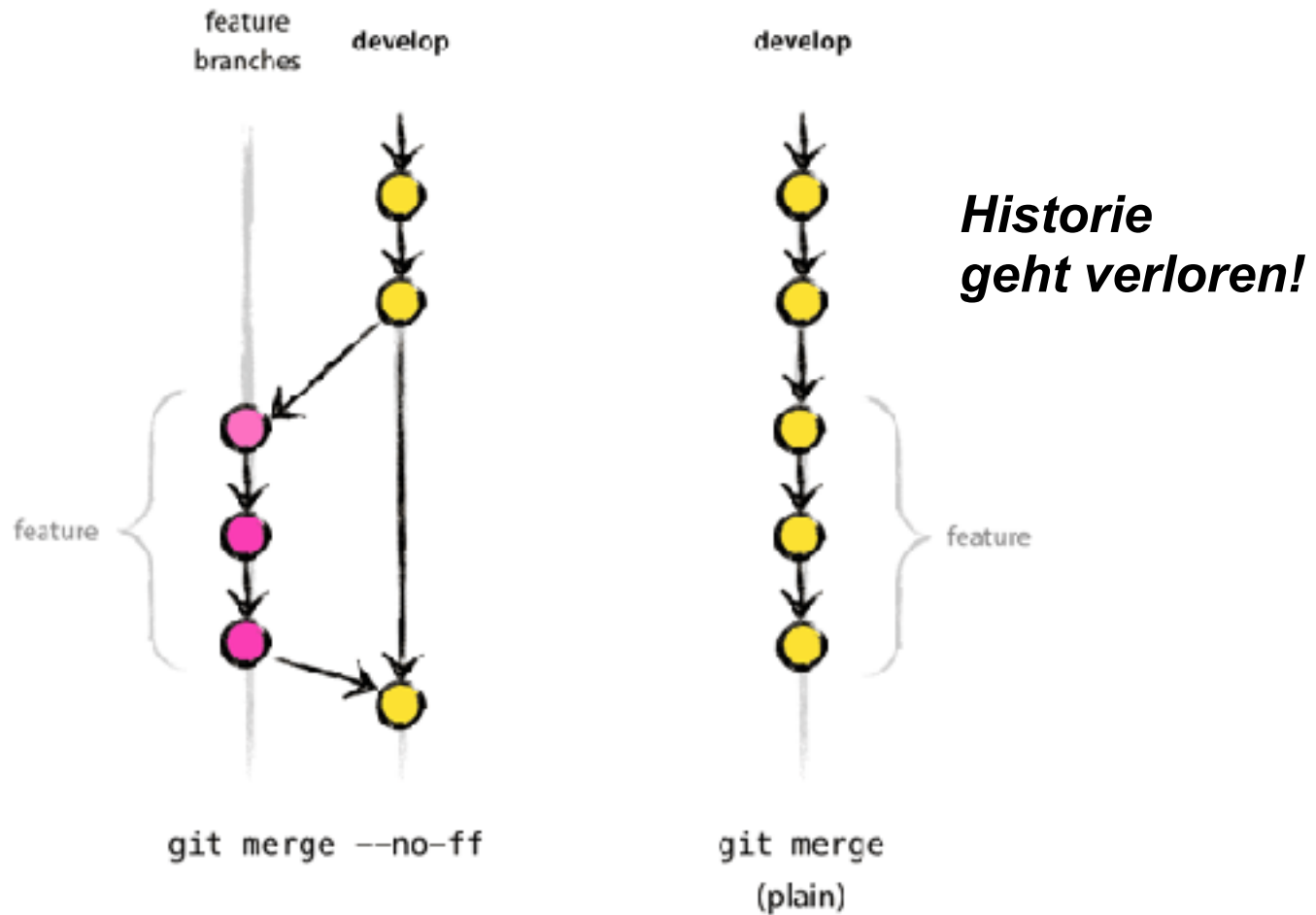
- Topic-branch mergen:

```
git checkout develop  
git merge --no-ff myTopic  
git branch -d myTopic # branch löschen  
git push origin develop # push to remote
```

feature  
branches      develop



## -no-ff Flag umgeht fast-forward merge



# Release branches

- Herkunft:
  - develop
- Wird gemerged in:
  - develop **und** master
- Namenskonvention:
  - *release-\**
- Idee: release-Branch wird gestartet, wenn Software quasi fertig ist für Release
- Auf dem release-Branch werden dann beispielsweise Versionsnummern in Artefakten usw. gesetzt, die auf develop-Branch nichts zu suchen haben
- Beispiel:

```
git checkout -b release-1.2 develop
./bump-version.sh 1.2          # eigenes Skript
git commit -a -m "Bumped version number to 1.2"
```



## Release branches (2)

- Herkunft:
  - develop
- Wird gemerged in:
  - develop **und** master
- Namenskonvention:
  - *release-\**
- Abschließen des release-Branches:

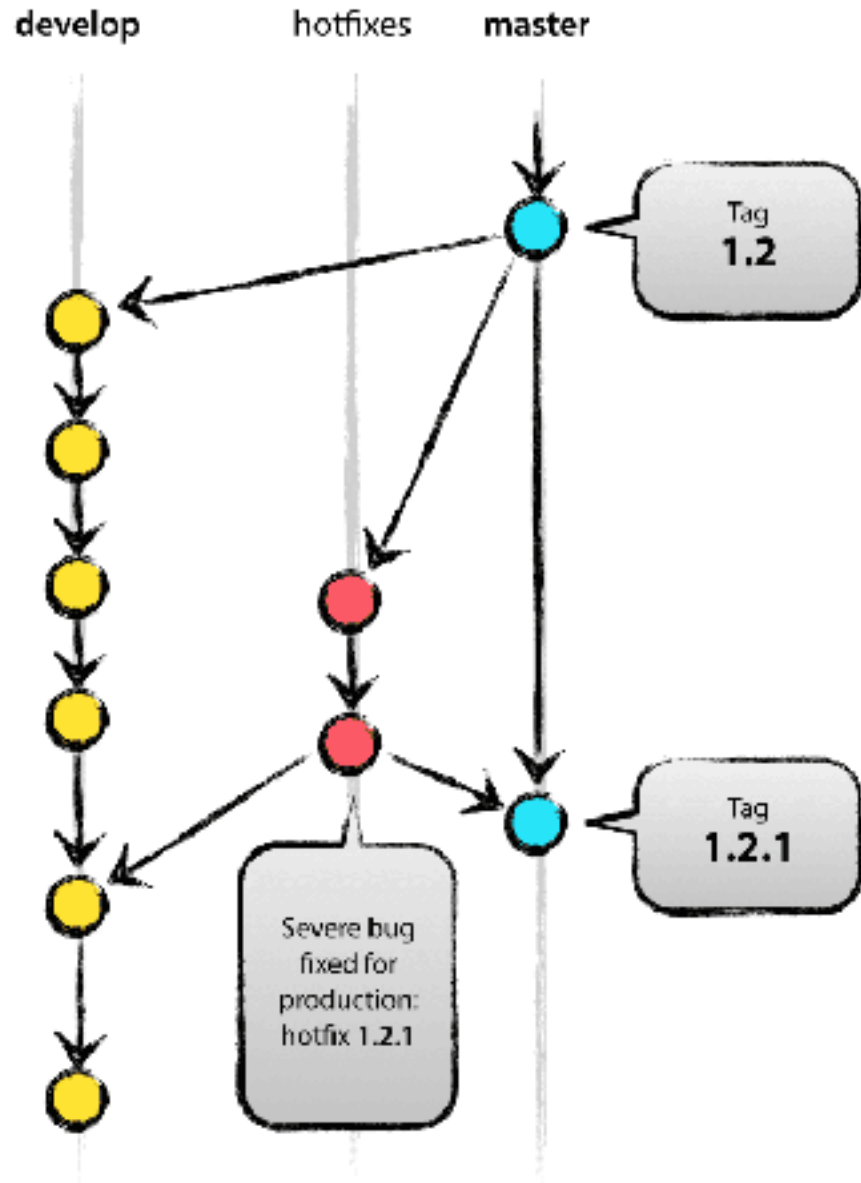
```
git checkout master  
git merge --no-ff release-1.2  
git tag -a 1.2
```

- Software ist nun releast. Nun ggf. noch Änderungen nach develop zurückspielen:

```
git checkout develop  
git merge --no-ff release-1.2
```

# Hotfix branches

- Herkunft:
  - master
- Wird gemerged in:
  - develop **und** master
- Namenskonvention:
  - *hotfix-\**
- Idee: Hotfix wird sofort in master eingespielt, aber zusätzlich in develop eingepflegt





<http://nvie.com/posts/a-successful-git-branching-model/>

# Git-Flow-Unterstützung

- Für die Kommandozeile:  
<https://github.com/nvie/gitflow/wiki/Installation>
- Durch GUI...
  - Empfehlung: SourceTree  
<https://www.sourcetreeapp.com/>

The screenshot shows the IntelliJ IDEA interface with the Git tool window open. The 'Merge pull request' button is highlighted with a red circle. The interface displays a commit history, branch list, and a diff view for a pull request.

**Git Tool Window:**

- Buttons:** View, Commit, Checkout, Revert, Stash, Add, Remove, Apply/Remove, Fetch, Pull, Push, Branch, Merge, Tag, Show in IDE, Git Flow, Terminal, Settings.
- File Status:** Working Copy.
- Branches:**
  - alternative-resolution-on-the-fly-10g
  - develop
  - feature
  - javaSupport
  - feature
  - indy-in-stable
  - master
  - mergeTest
  - release-1
- Tags:**
- Remotes:**
  - origin
    - 1.0.1.dev.3
    - 1.2.1.dev.17
    - 1.2.2.dev.38
    - 1.2.3.alpha.1
    - 1.2.3.parallel.11
    - 1.2.3.dev.8
    - 1.beta.3.dev.16.patrick.1
    - 1.beta.3.patrick-merged.1
    - 1.beta.3.patrick.1
    - 1.beta.4.dev.27.patrick.1
    - 1.beta.4.dev.33.patrick.1
    - 1.beta.4.dev.35.patrick.1
    - 1.beta.4.dev.36.patrick.1
    - 1.beta.4.dev.37.patrick.1
    - 1.beta.4.dev.40.patrick.1
    - 1.beta.4.dev.46.patrick.1
    - 1.beta.4.dev.50.patrick.1
    - 1.beta.4.dev.53.patrick.1
    - 1.beta.4.dev.54.patrick.1
    - 1.beta.4.dev.55.patrick.1
    - 1.beta.4.dev.56.patrick.1
    - 1.beta.4.dev.57.patrick.1
    - 1.beta.4.dev.58.patrick.1
    - 1.beta.4.dev.59.patrick.1
    - 1.beta.4.dev.60.patrick.1
    - 1.beta.4.dev.61.patrick.1
    - 1.beta.4.dev.62.patrick.1
    - 1.beta.4.dev.63.patrick.1
    - 1.beta.4.dev.64.patrick.1
    - 1.beta.4.dev.65.patrick.1
    - 1.beta.4.dev.66.patrick.1
    - 1.beta.4.dev.67.patrick.1
    - 1.beta.4.dev.68.patrick.1
    - 1.beta.4.dev.69.patrick.1
    - 1.beta.4.dev.70.patrick.1
    - 1.beta.4.dev.71.patrick.1
    - 1.beta.4.dev.72.patrick.1
    - 1.beta.4.dev.73.patrick.1
    - 1.beta.4.dev.74.patrick.1
    - 1.beta.4.dev.75.patrick.1
    - 1.beta.4.dev.76.patrick.1
    - 1.beta.4.dev.77.patrick.1
    - 1.beta.4.dev.78.patrick.1
    - 1.beta.4.dev.79.patrick.1
    - 1.beta.4.dev.80.patrick.1
    - 1.beta.4.dev.81.patrick.1
    - 1.beta.4.dev.82.patrick.1
    - 1.beta.4.dev.83.patrick.1
    - 1.beta.4.dev.84.patrick.1
    - 1.beta.4.dev.85.patrick.1
    - 1.beta.4.dev.86.patrick.1
    - 1.beta.4.dev.87.patrick.1
    - 1.beta.4.dev.88.patrick.1
    - 1.beta.4.dev.89.patrick.1
    - 1.beta.4.dev.90.patrick.1
    - 1.beta.4.dev.91.patrick.1
    - 1.beta.4.dev.92.patrick.1
    - 1.beta.4.dev.93.patrick.1
    - 1.beta.4.dev.94.patrick.1
    - 1.beta.4.dev.95.patrick.1
    - 1.beta.4.dev.96.patrick.1
    - 1.beta.4.dev.97.patrick.1
    - 1.beta.4.dev.98.patrick.1
    - 1.beta.4.dev.99.patrick.1
    - 1.beta.4.dev.100.patrick.1

**Commit History:**

Graph	Description	Commit	Author	Date
[1] origin/develop	Merge pull request #559 from MaroM/feature-WriteAnnotations	fb425f6	Steven Arzt <Steven.Arzt@casad.de>	23.03.2016, 21:18
[2] origin/HEAD	Update AnnotationEnumElem.java	8447e45	MaroM <maro@casad.de>	23.03.2016, 21:18
[3] origin/develop	fixes a race condition	666a56a	Steven Arzt <Steven.Arzt@casad.de>	23.03.2016, 17:07
[4] origin/develop	fixes a race condition	d992d3b	Steven Arzt <Steven.Arzt@casad.de>	11.03.2016, 14:15
[5] origin/develop	fixes a race condition	d6a426f	Steven Arzt <Steven.Arzt@casad.de>	11.03.2016, 13:07
[6] origin/develop	Merge pull request #559 from MaroM/feature-locals	fb425f6	Steven Arzt <Steven.Arzt@casad.de>	10.03.2016, 09:27
[7] origin/develop	Adds the ability to write out local variable names using the ASM backend. In order to...	fb425f6	MaroM <maro@casad.de>	10.03.2016, 09:02
[8] origin/develop	fixes a race condition	fb425f6	Steven Arzt <Steven.Arzt@casad.de>	01.03.2016, 16:01
[9] origin/develop	[1] origin/CILFromEnd - simplifies the whole process stuff to not create derived classes	fb425f6	Steven Arzt <Steven.Arzt@casad.de>	21.03.2016, 11:08
[10] origin/develop	refactorings and some more work on generics	fb425f6	Steven Arzt <Steven.Arzt@casad.de>	11.03.2016, 11:09
[11] origin/develop	moved the trap data structure into the ast package	fb425f6	Steven Arzt <Steven.Arzt@casad.de>	09.03.2016, 16:21

**Diff View:**

src\scotitagkit\AnnotationEnumElem.java

Hunk 1: Lines 48-50

```

48  +  @Override
49  +  public String toString() {
50  +  }

```

Hunk 2: Lines 57-62

```

57  +  public void setTypeName(String newValue) {
58  +  typeName = newValue;
59  +  }
60  +  }
61  +  }
62  +  public String getConstantName() {

```

Hunk 3: Lines 65-70

```

65  +  public void setConstantName(String newValue) {
66  +  constantName = newValue;
67  +  }
68  +  }
69  +  }
70  +  }

```

**Commit Details:**

Commit: fb425f6e805e1d5e2e2b1f50f1db106c8a (fb425f6)  
 Parents: 666a56a56a, 8447e45b3a  
 Author: Steven Arzt <Steven.Arzt@casad.de>  
 Date: 23. März 2016 um 21:18:27 MEZ  
 Labels: HEAD origin/develop origin/HEAD release/release-1 develop

**Merge pull request #559 from MaroM/feature-WriteAnnotations**

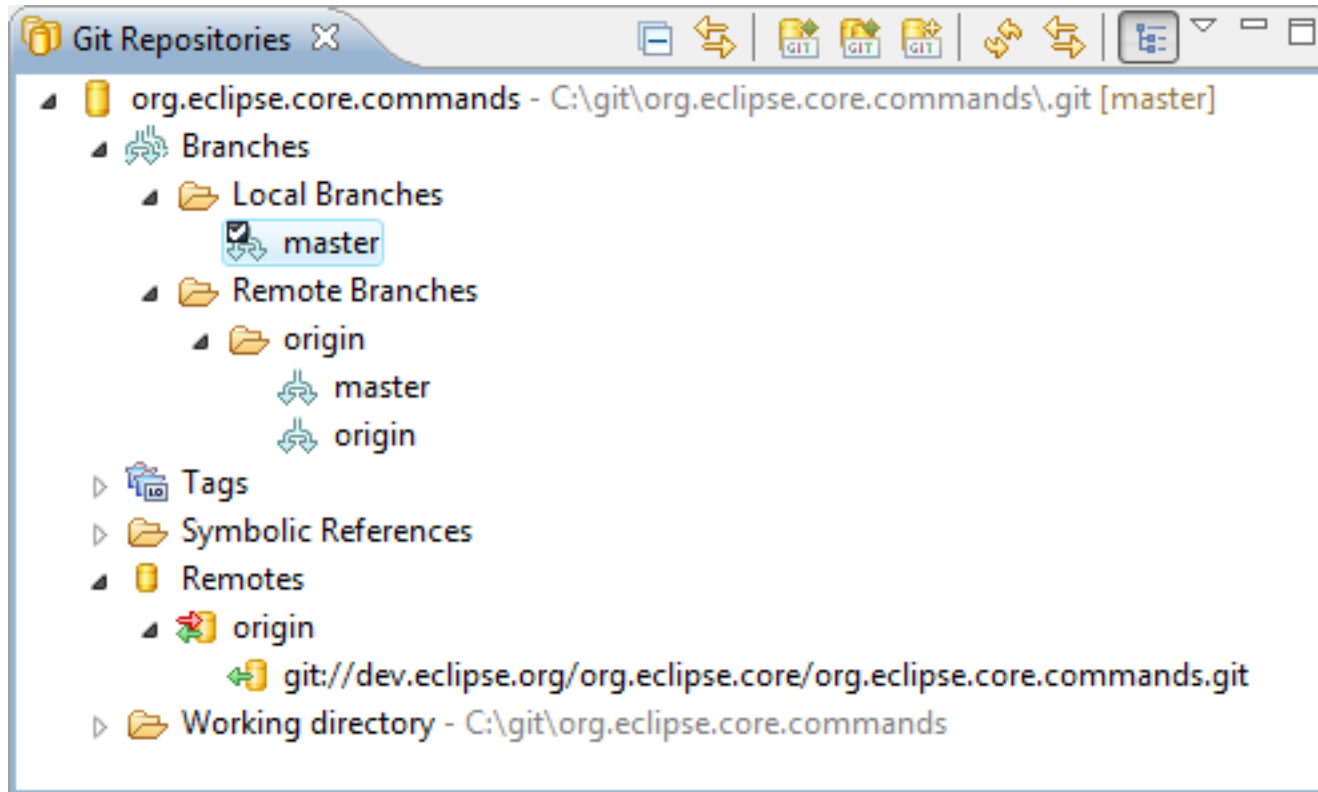
## Atlassian

The screenshot displays the IntelliJ IDEA IDE with the following components:

- Top Toolbar:** Includes icons for View, Commit, Checkout, Revert, Stash, Add, Remove, Add/Remove, Fetch, Pull, Push, Branch, Merge, Tag, Show in Finder, CI Flow, and Tooling.
- Left Sidebar:**
  - FILE STATUS:** Shows Working Copy.
  - BRANCHES:** Lists branches like alternative-annotation-on-the-fly-idea, develop, feature, javaSupport, testenv, test-to-static, master, merged, release, and release-1.
  - TAGS:** Empty list.
  - REMOTES:** Lists remotes like 1.0.1.dev.3, 1.2.1.dev.17, 1.2.2.dev.38, 1.2.2.for.unspx.2, 1.2.2.pointer.11, 1.2.b.dev.8, 1.beta.3.dev.10.patrick.1, 1.beta.3.patrick-merged.1, 1.beta.3.patrick.1, 1.beta.4.dev.27.patrick.1, 1.beta.4.dev.33.patrick.1, 1.beta.4.dev.33.phantom.1, 1.beta.4.dev.35.phantom.1, 1.beta.4.dev.42.patrick.1, 1.beta.4.dev.45.patrick.2, 1.beta.4.dev.58.patrick.1, 1.beta.4.dev.63.7.2, 1.beta.4.dev.71.patrick.1, 1.beta.4.dev.75.java-to-idea.2, 1.beta.4.patrick.1, 1.beta.5.dev.18.patrick.1, and 1.beta.5.dev.75.patrick.1.
- Main Editor:**
  - Git Branches:** A graph showing the development branch and a new branch named 'myFeature'.
  - Commit Dialog:** A modal window for creating a new branch. It shows 'Feature Name: myFeature', 'Start at: Latest development branch', and a preview of the branch structure.
  - Commit History:** A list of recent commits. The selected commit is 'Commit: fb425184e0605ef6e64e2c8b6c50c1db168c5e (fb42518)' by 'Steven Arzt' on '23. März 2016 um 21:21:27 MEZ'. It includes a small profile picture of Steven Arzt.
  - Code Diff:** A view showing the changes in the file 'src/soot/tagkit/AnnotationEnumElem.java'. It highlights three hunks: 'Hunk 1: Lines 48-50', 'Hunk 2: Lines 67-68', and 'Hunk 3: Lines 65-70'.
- Right Sidebar:**
  - Author Table:** A table with columns 'Author' and 'Date'. It lists several commits by 'Steven Arzt' and 'Marc Miltenberger'.
  - Search:** A search bar with the text 'Search'.
- Bottom Bar:** Shows the 'Atlassian' logo.

# EGit

- EGit integriert Git-Unterstützung in Eclipse



- <http://www.eclipse.org/egit/>



# Ein Repository für alles nutzen?

- Wir empfehlen in der Tat, sämtlichen Code und andere Artefakte in einem einzigen Repository zu versionieren. Warum?
  - Man kann einfacher identifizieren, wann Fehler eingeführt wurden?
    - Beispielsweise operieren Befehle wie `git bisect` stets auf einem einzigen Repository. Dieser Befehl erlaubt automatisches Delta-Debugging, also das Auffinden des für eine Regression verantwortlichen Commits, mittels Binärsuche.
    - Die Nachverfolgung der Historie eines gegebenen Features wird stark vereinfacht.
    - Man kann Dokumente wie Dokumentationen, (automatisch erstellte?) Testprotokolle usw. direkt zur passenden Code-Version ablegen
    - Das gesamte Management der Versionierung wird vereinfacht und kostet weniger Zeit und Aufwand.

# Zusammenfassung

- Es gibt Systeme zur Zentralen und Dezentralen Versionierung
- Die Dezentrale Versionierung ist etwas komplexer, bietet jedoch viele Vorteile
- Git unterstützt die dezentrale Versionierung
- Git-Flow gibt hilfreiche Konventionen zum Management von Branches
- Werkzeuge wie SourceTree unterstützen diese durch Automatisierung
- Nähere Informationen zum Nachlesen: Pro Git Ebook  
<https://git-scm.com/book/en/v2>

***Weiter geht's morgen 18:00 s.t. (!!!)***

