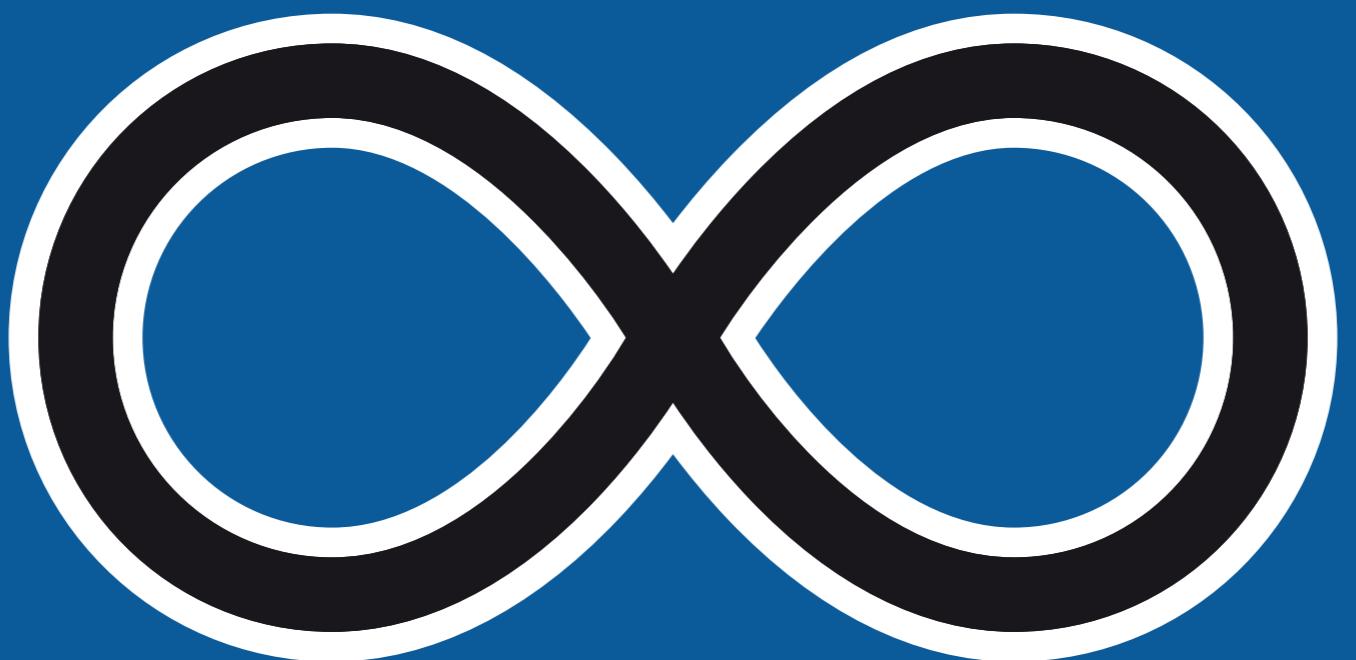


SOFTWAREENTWICKLUNG

IM TEAM MIT OPEN-SOURCE-WERKZEUGEN

04 - verteiltes Versionsmanagement im Team



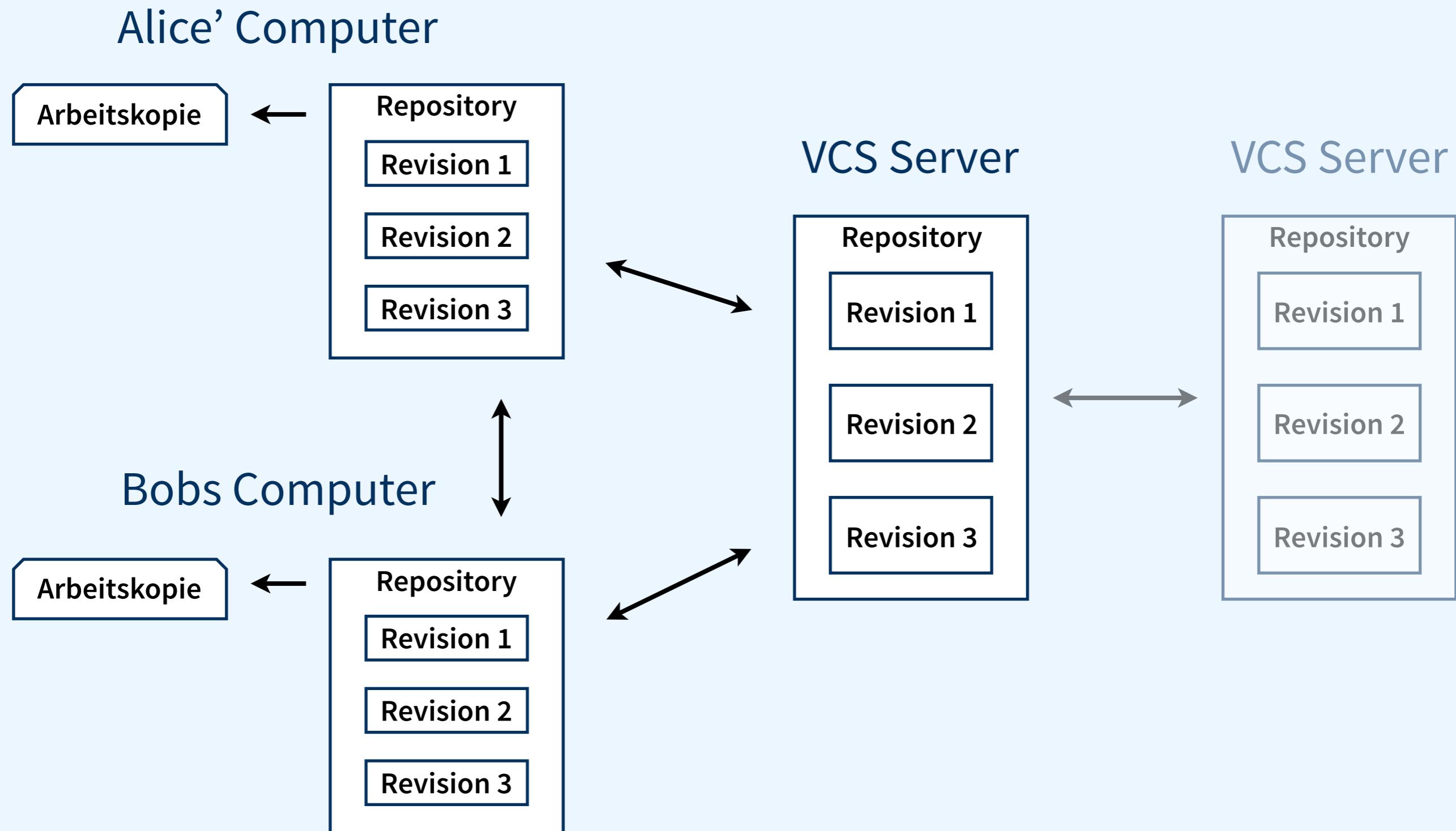
WIEDERHOLUNG

verteilte Versionskontrolle

Lösung:

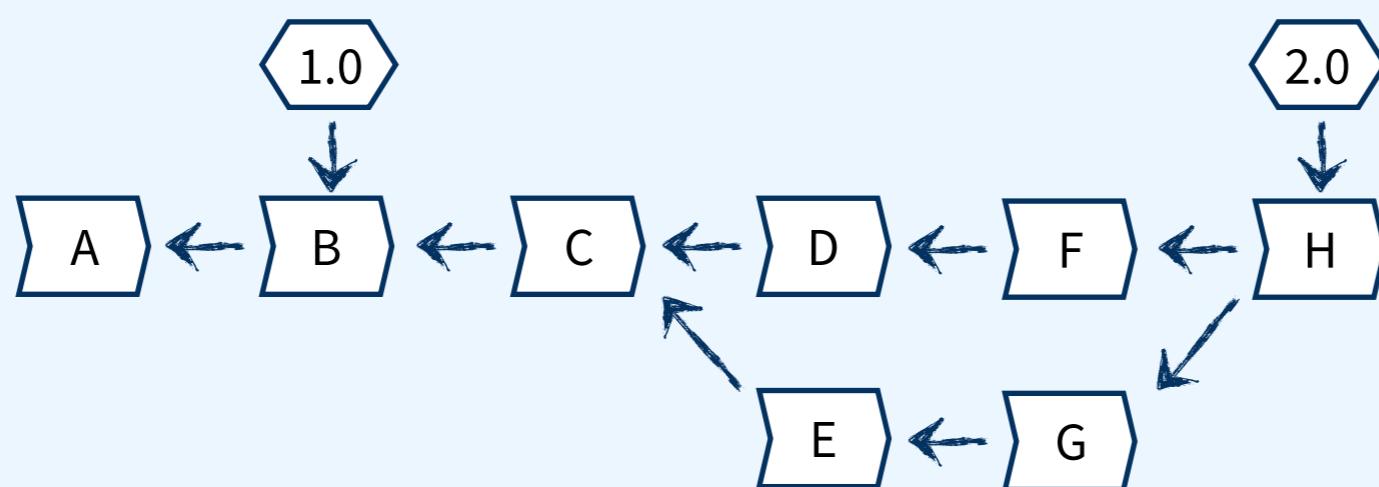
- zentrales VCS durch verteiltes VCS ersetzen
- Nutzer holen sich keine Arbeitskopie
 - ▶ sondern eine Kopie des gesamten Repository
- Nutzer kann auch ohne Netzwerkverbindung arbeiten
- Wenn der Server komplett ausfällt
 - ▶ Repository von einem Nutzer wieder auf den Server kopieren
- Hierarchische Repository-Strukturen sind möglich!

verteilte Versionskontrolle



Commit-Graph

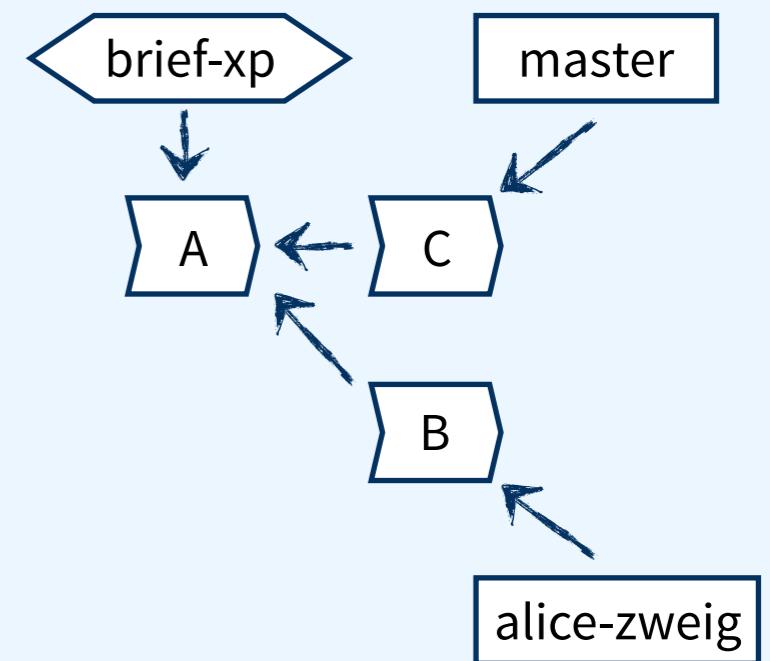
- Der Graph der Objektdatenbank lässt sich auf die Commits, Tags und Branches reduzieren
- Wird »Commit-Graph« genannt
- Git kennt keine Revisionsnummern, nur Hashwerte der Commits
- Es entsteht ein gerichteter azyklischer Graph (DAG)
- GUI-Tools können diesen Graphen darstellen



Branch erzeugen

- Mit »git branch [zweig]« wird ein neuer Zweig erstellt
 - ▶ neues Branch-Objekt im Repository erzeugt
- Mit »git checkout [zweig]« wird auf den neuen Zweig gewechselt
 - ▶ bestimmt, in welchen Branch Commits geschrieben werden

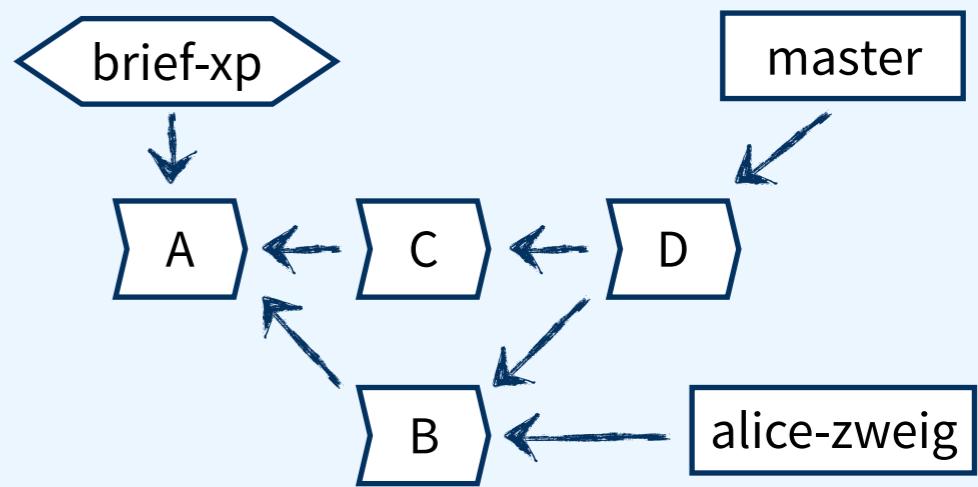
```
/GitProjekt $ git branch alice-zweig
/GitProjekt $ git checkout alice-zweig
Switched to branch 'alice-zweig'
/GitProjekt $ emacs brief.txt
/GitProjekt $ git commit -a -m "Änderungen im Zweig"
[alice-zweig e0a0374] Änderungen im Zweig
 1 file changed, 1 insertion(+)
/GitProjekt $ git checkout master
Switched to branch 'master'
/GitProjekt $ emacs brief.txt
/GitProjekt $ git commit -a -m "Änderungen im Master"
[master 8436b95] Änderungen im Master
 1 file changed, 1 insertion(+), 1 deletion(-)
```

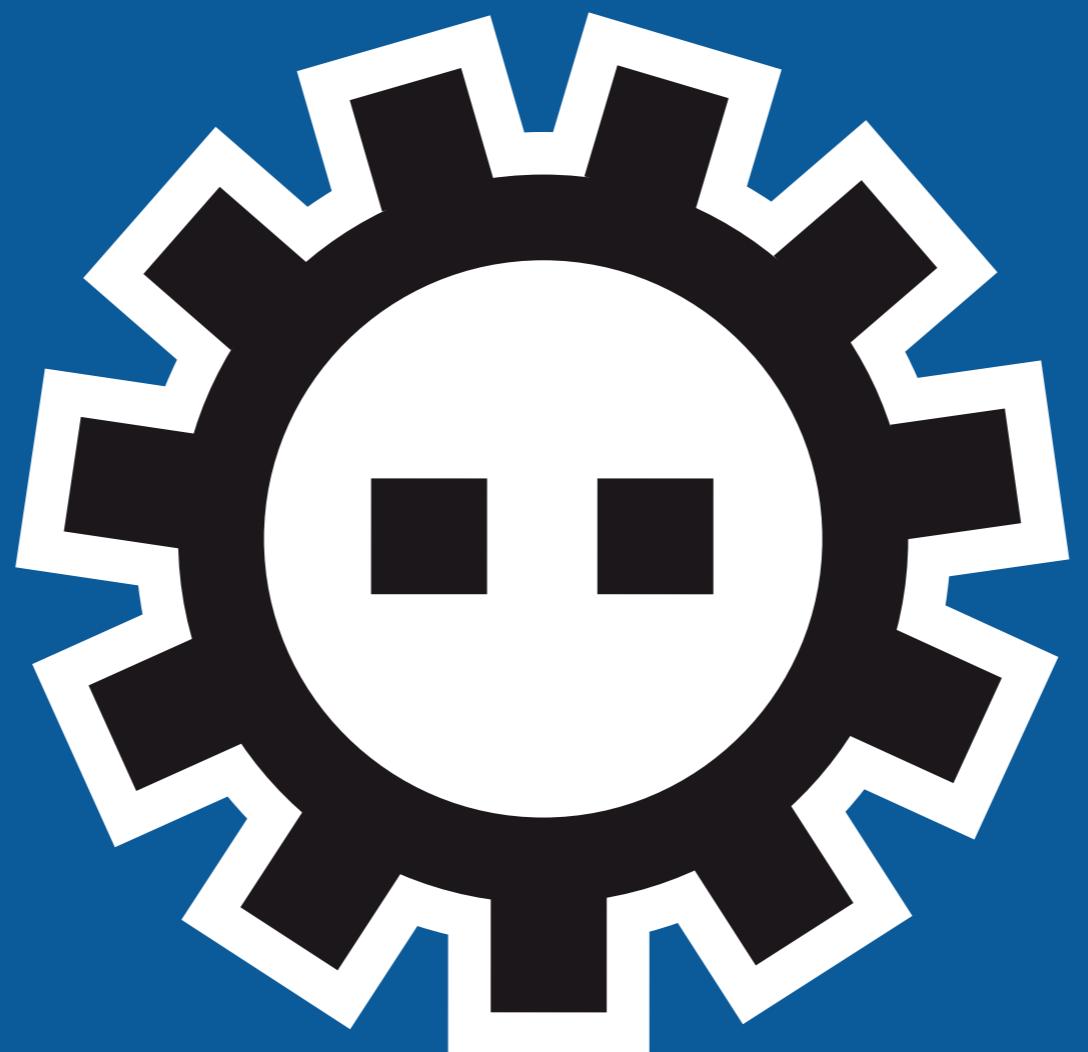


Branch Zusammenführen

- Das Zusammenführen (Merge) läuft ähnlich wie in Subversion:
- Zielzweig als Arbeitskopie auschecken
- »git merge [zweig]« mit dem Quellzweig aufrufen
 - ▶ Änderungen werden automatisch committed
- Eventuell Quellzweig löschen (entspricht löschen eines Zeigers)
 - ▶ mit »git branch -d [zweig]«

```
/GitProjekt $ git branch
* master
  alice-zweig
/GitProjekt $ git merge alice-zweig
Auto-merging brief.txt
Merge made by the 'recursive' strategy.
  brief.txt | 1 +
  1 file changed, 1 insertion(+)
```





MOTIVATION

Git im Team

- Technisch ist ein reines peer-to-peer möglich:
 - ▶ jeder Entwickler hat eigenes Repository
 - ▶ Austausch nur zwischen den Entwicklern
- Die offenen Fragen bleiben:
 - ▶ Wer arbeitet aber an welchen Änderungen?
 - ▶ Wie komme ich an Änderungen, wenn der Rechner des anderen Entwicklers ausgeschaltet ist?
- Besser einen Server mit einem Git-Repository bereitstellen:
 - ▶ zentrale, immer erreichbare Instanz mit Datensicherung
 - ▶ trotzdem können Entwickler dezentral arbeiten

Git Server

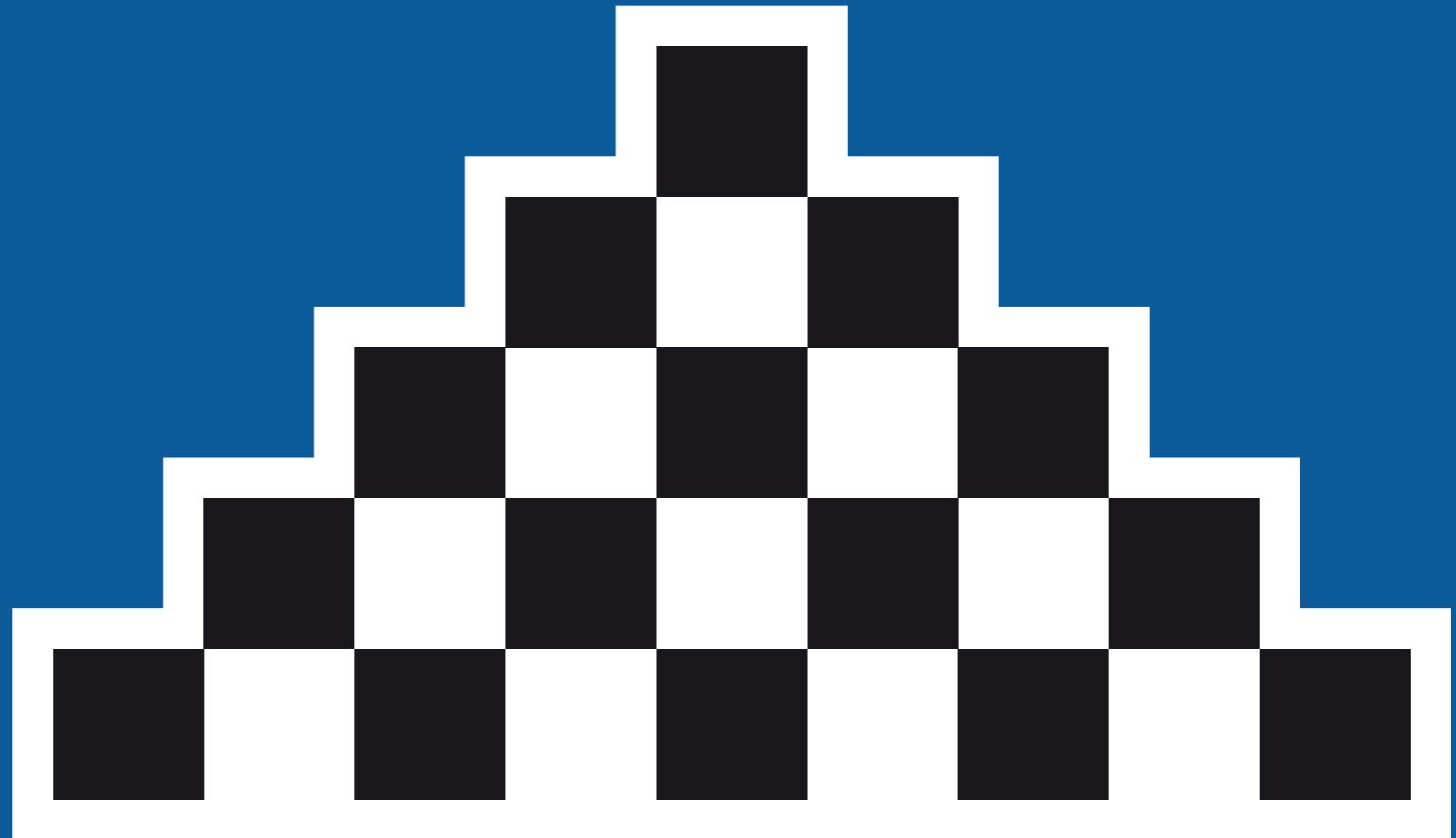
Zwei Möglichkeiten für zentrale Git-Repository:

Eigener Git-Server

- Repository ohne Arbeitskopie
 - ▶ das .git-Verzeichnis
- Verschiedene Zugriffsarten:
 - ▶ file://
 - ▶ ssh://
 - ▶ git://
 - ▶ http://

Gehosteter Git-Server

- Anbieter im Internet
 - ▶ kostenlos für öffentliche Repository
- Web-Oberfläche zur Administration
- »soziale« Funktionen
- GitHub, BitBucket, Google, Microsoft, SourceForge

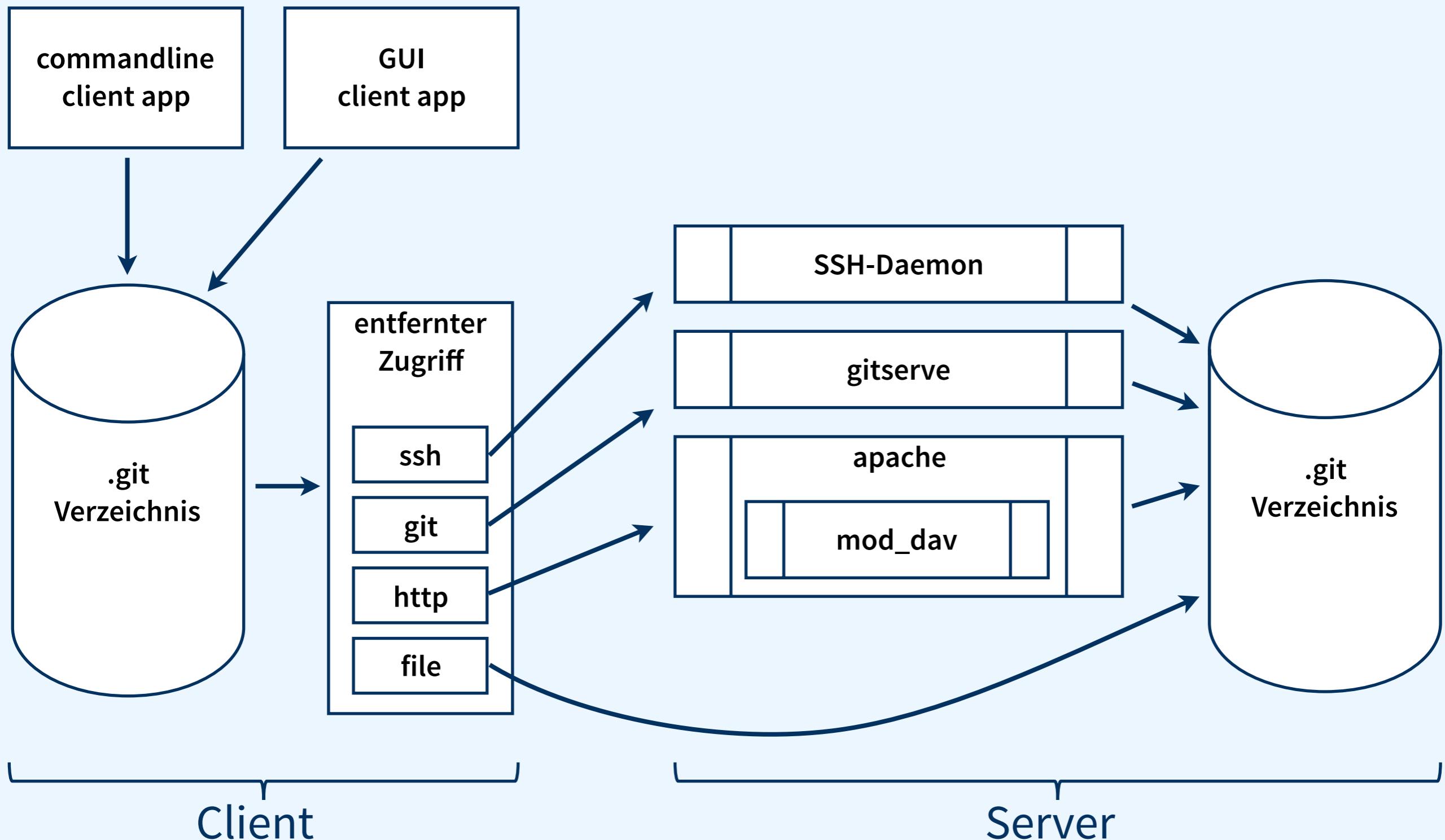


GRUNDLAGEN

Zugriffsarten

- Das Git-Repository ist nur ein Ordner im Dateisystem
 - ▶ das .git Verzeichnis
- Der entfernte Zugriff auf das Dateisystem ist eine generische Anforderung
 - ▶ Git erfindet nicht das Rad neu
 - ▶ überlässt die Zugriffsverwaltung etablierten Technologien
- Verschiedene Technologien für verschiedene Szenarien:
 - ▶ nur lesend oder auch schreibend?
 - ▶ geschlossene Benutzergruppe oder anonym zugänglich?
 - ▶ große oder kleine Teams?

Architektur



Zugriffsart: File

Das externe Repository ist in einem anderen Verzeichnis auf der Festplatte

- z.B. ein mittels NFS oder SMB eingebundenes Netzlaufwerk
 - ▶ aus Sicht von Git ein lokaler Pfad

Vorteile:

- schnellster Zugriff und einfachste Einrichtung

Nachteile:

- Netzfreigaben sind häufig nur für lokale Netze ausgelegt
 - ▶ Probleme beim Zugriff aus dem Internet

Zugriffsart: SSH

Das externe Repository wird über einen SSH-Zugang (ver-schlüsselte Shell) angesprochen

- Lässt sich leicht auf Servern einrichten
- Erlaubt auch Schreiboperationen auf das Repository

Vorteile:

- Komplett verschlüsselter und authentisierter Zugang
- Datentransfer wird komprimiert

Nachteile:

- Kein anonymer Zugriff, alle Nutzer müssen sich authentisieren

Zugriffsart: GIT

Das externe Repository wird über ein Git-eigenes Server-programm veröffentlicht

- Sehr einfaches proprietäres Protokoll

Vorteile:

- Kompaktestes Übertragungsprotokoll, ohne Overhead für Verschlüsselung und Authentisierung

Nachteile:

- Da keine Authentisierung:
 - ▶ entweder hat niemand Schreibrechte
 - ▶ oder alle haben Schreibrechte
- Kein Standard-Port (9418) → häufig per Firewall gesperrt

Zugriffsart: HTTP

Das externe Repository wird über einen Web-Server nach außen veröffentlicht

- Das .git Verzeichnis muss dafür nur als statisches Verzeichnis im htdocs-Ordner liegen

Vorteile:

- Standard-Dienst, geht durch fast alle Firewalls
- Weltweiter, anonymer Zugriff

Nachteile:

- Viel Overhead bei der Kommunikation
- Schreibzugriff nur mittels komplexer Webdav-Konfiguration

Fazit Zugriffsart

- Obwohl Git viele Zugriffsarten kennt, hat sich folgende Struktur als pragmatisch erwiesen:
- Das Entwicklerteam verwendet individuelle Schlüssel um mittels SSH lesend und schreibend auf das Repository zuzugreifen
- Falls ein anonymer Zugang aus dem Internet möglich sein soll:
 - ▶ das gleiche Repository wird auch über HTTP veröffentlicht
- Wie werden die Benutzer (Entwicklerteam) verwaltet?

Benutzerverwaltung

- **File:** Dateizugriffsrechte des Dateisystems
- **SSH:** SSH-Zugang (meist Benutzerkonto auf dem Rechner)
- **HTTP:** Benutzerdatenbank des Web-Servers
- **GIT:** Keine Benutzerverwaltung
- Einheitlicher Zugang über alle Protokolle ist schwer zu realisieren
- Häufig wird nur der Zugang mittels SSH und HTTP angeboten
- Üblich ist der Aufbau (bzw. Nutzung) eines LDAP-Verzeichnisses zur Benutzerverwaltung
 - ▶ die ersten drei Zugriffsarten können darauf zurückgreifen



WERKZEUGE

Zugriff auf Git im Team

Git auf dem Server

Herunterladen eines Repos

- Das Herunterladen eines entfernten Repositories wird klonen genannt
 - ▶ `git clone [REPO-URL]` erzeugt ein lokales Repository mit einer exakten Kopie des entfernten Repositories
 - ▶ zusätzlich wird die Ursprungs-URL der Kopie lokal gespeichert (unter dem Namen `origin`)
- Zwei weitere Funktionen stehen auf geklonten Repositories bereit:
 - ▶ `git pull` holt alle neuen Commits vom entfernten Repository und pflegt sie in das lokale Repository ein
 - ▶ `git push` alle lokalen Commits, die noch nicht im entfernten Repository bekannt sind, werden veröffentlicht

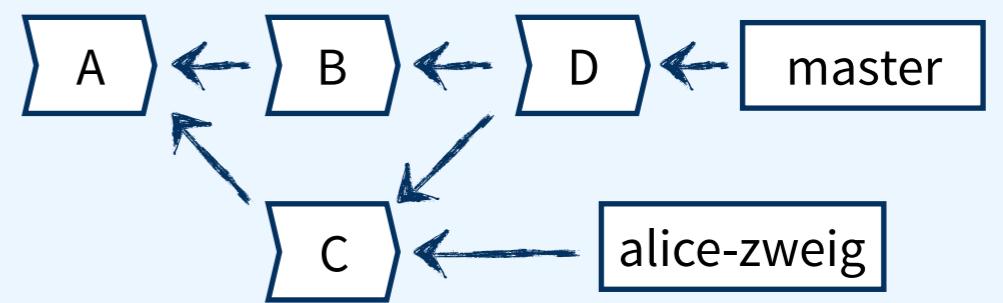
Clone - Pull - Push

```
/ $ git clone https://github.com/betermieux/  
GitProjekt.git  
Cloning into 'GitProjekt'...  
remote: Reusing existing pack: 16, done.  
remote: Total 16 (delta 0), reused 0 (delta 0)  
Unpacking objects: 100% (16/16), done.  
Checking connectivity... done.  
/ $ cd GitProjekt  
/GitProjekt $ git pull  
Updating 24a74e1..fd1c012  
Fast-forward  
.project | 11 ++++++++  
brief.txt | 2 +-  
2 files changed, 12 insertions(+), 1 deletion(-)  
/GitProjekt $ emacs brief.txt  
/GitProjekt $ git push  
Counting objects: 7, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 318 bytes | 0 bytes/  
s, done.  
Total 3 (delta 1), reused 0 (delta 0)  
To https://github.com/betermieux/GitProjekt.git  
7238a96..fd1c012 master -> master
```

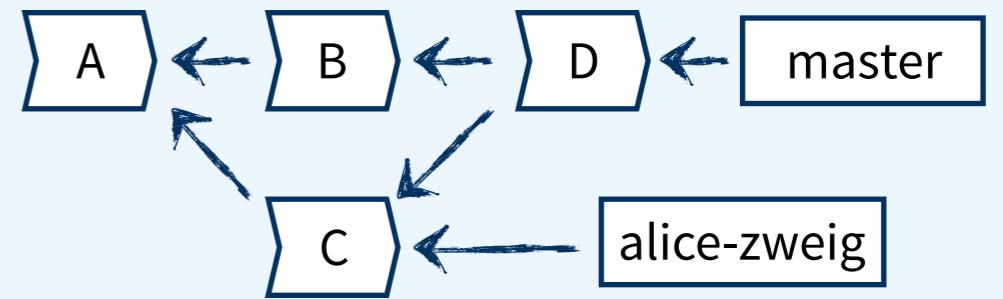
Lokal: Nach »git clone«



Server: Nach einigen Commits anderer Nutzer

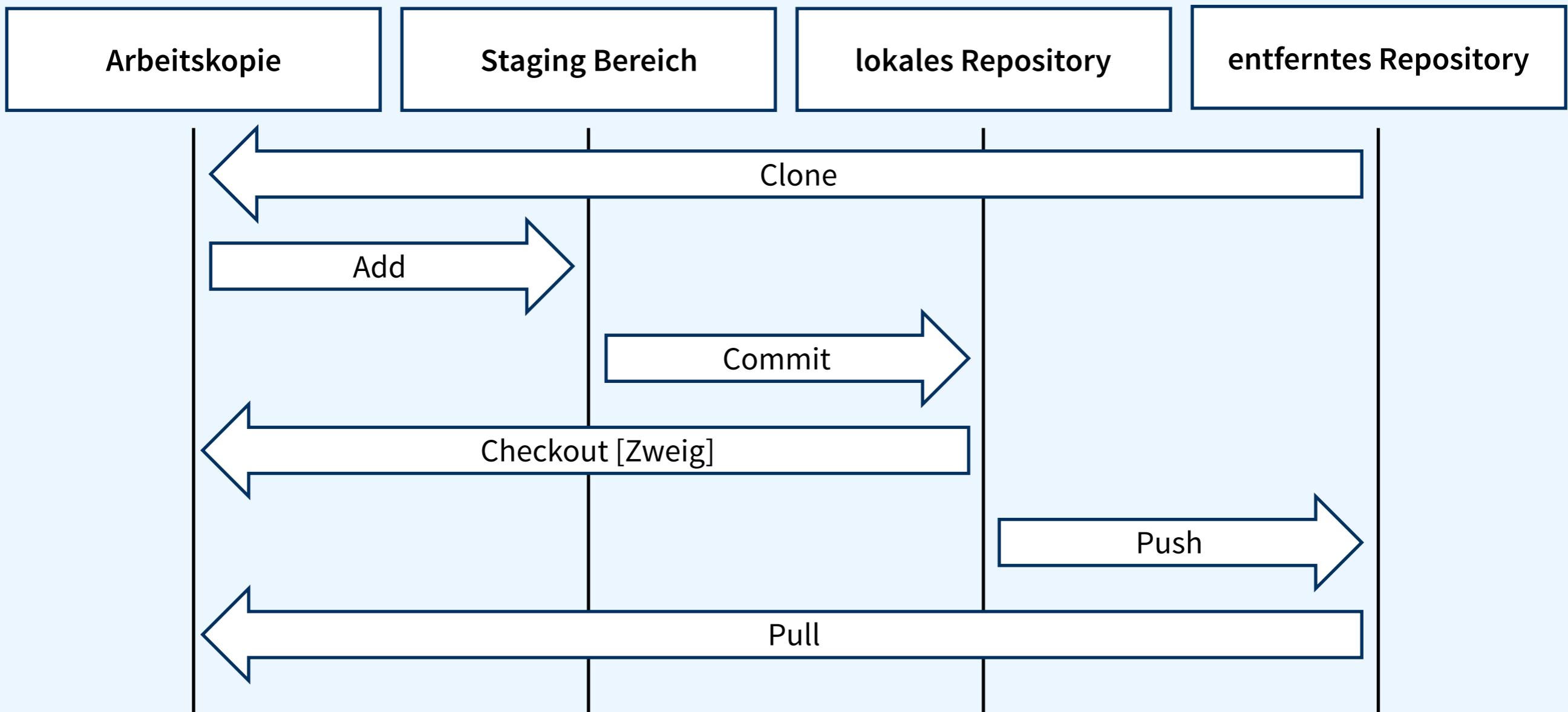


Lokal: Nach »git pull«



Kommunikation

Übersicht über die Auswirkung von Git-Operationen:



Git Hosting und Kollaboration

Github

GitHub

- Webbasiertes Git-Hosting
 - ▶ kostenlos für Open-Source-Projekte
- Forks(clones) und Merges werden propagiert
 - ▶ aufgrund der hash-basierten Speicherung in Git können tausende von Forks eines Projekts verwaltet werden
- Grafische Ansichten für:
 - ▶ Entwickleraktivitäten
 - ▶ Branch-Merge-Baum
- »Soziale« Funktionen:
 - ▶ Nutzer folgen (follow)
 - ▶ Projekte empfehlen (star)

GitHub Verbreitung

- GitHub wird von vielen Open-Source-Projekten als zentrale Entwicklungsinstantz verwendet, z.B.:

- ▶ Git
- ▶ jQuery
- ▶ JUnit
- ▶ Perl
- ▶ Ruby on Rails
- ▶ Joomla
- ▶ NodeJS



GitHub Demo

The screenshot shows a GitHub repository page for 'betermieux / GitProjekt'. The top navigation bar includes links for 'Explore', 'Gist', 'Blog', and 'Help'. The user profile 'betermieux' is visible on the right. The repository name 'GitProjekt' is displayed, along with options to 'Unwatch' (1), 'Star' (0), or 'Fork' (0). The 'Description' section contains a text input field for a short description of the repository. The 'Website' section has a text input field for a website URL and a 'Save' button. Below these are summary statistics: 5 commits, 2 branches, 1 release, and 1 contributor. A green 'Code' button is present. The main content area shows a commit history with three entries:

- betermieux authored 5 months ago (latest commit)
- .project added project 5 months ago
- brief.txt Merge branch 'alice-zweig' 5 months ago

A note at the bottom encourages adding a README, with a blue 'Add a README' button.

On the right side, there is a sidebar with links to 'Issues' (0), 'Pull Requests' (0), 'Wiki', 'Pulse', 'Graphs', 'Network', and 'Settings'. The 'Code' link is currently selected.

HFU Prof.Dr.Stefan Betermieux | Fakultät Informatik | Hochschule Furtwangen

Mitarbeit an Open-Source-Projekten

Github Workflow

Motivation

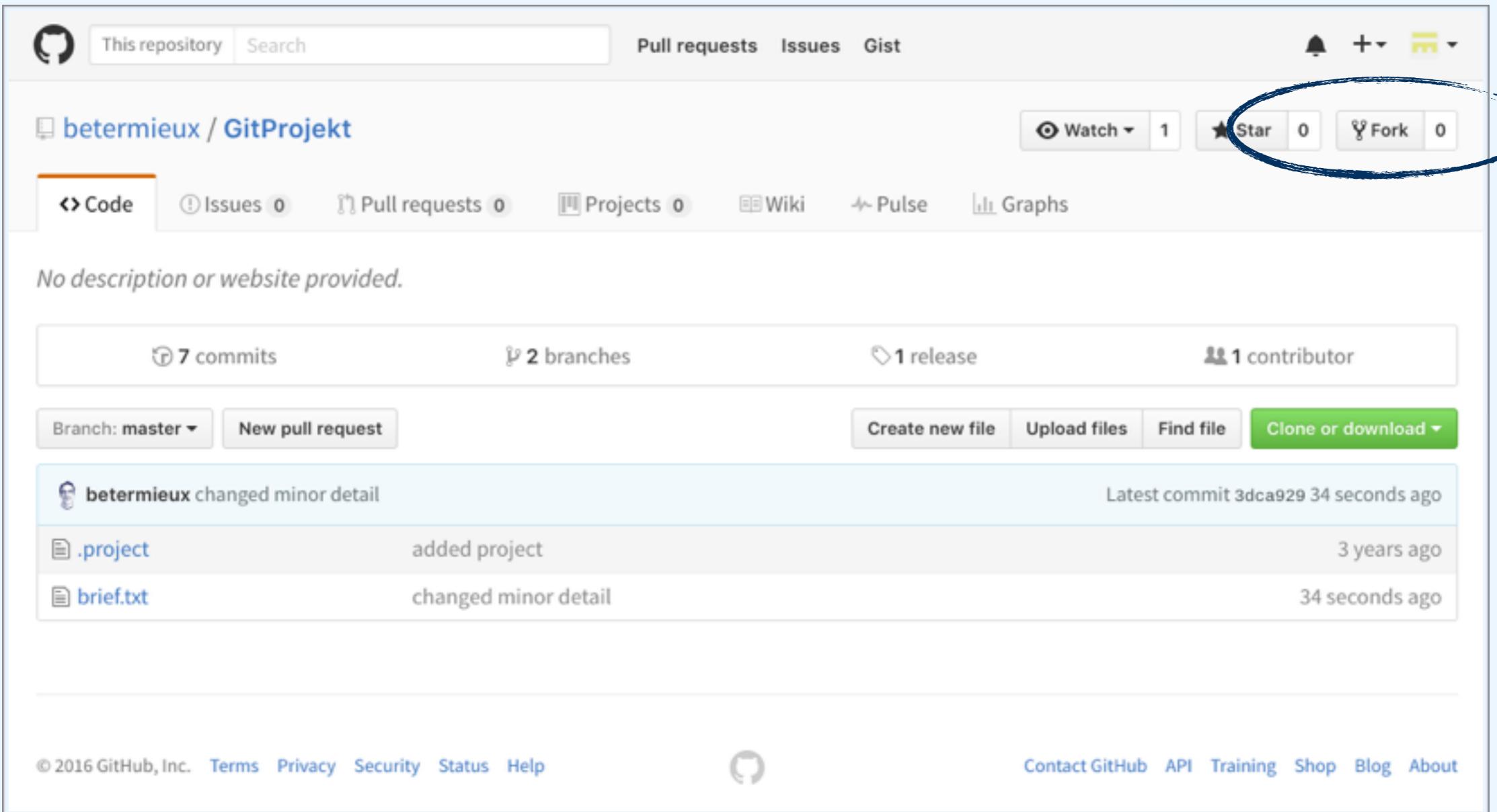
- Wie kommen eigene Änderungen in fremde Repositories
 - ▶ meist hat man keine Schreibrechte
- Besitzer des Ursprungs-Repository eine Nachricht schicken
 - ▶ mit der Bitte, Änderungen aus einem eigenen Repository zu holen und selbst einzupflegen
 - ▶ der Besitzer des Ursprungs-Repository kann die Änderung überprüfen (Qualitätskontrolle)
- Diese Nachricht wird Pull-Request genannt

Github Workflow

1. Ursprungsprojekt *forken*
2. Einen eigenen *Branch* vom *master* erstellen
3. Eigene *Commits* in diesen *Branch* erstellen
4. Alle *Commits* aus diesem *Branch* in das geforkte Projekt auf Github *pushen*
5. Einen *Pull-Request* im Ursprungsprojekt öffnen
6. Diskussionen führen und evtl. weitere *Commits* durchführen
7. Der Besitzer des Ursprungsprojekts führt einen *Merge* durch

1: Ursprungsprojekt forken

- Beliebiges öffentliches Projekt aufrufen
- Als angemeldeter Nutzer den Fork Knopf drücken



The screenshot shows a GitHub repository page for 'betermieux / GitProjekt'. The top navigation bar includes 'This repository', 'Search', 'Pull requests', 'Issues', 'Gist', and user notifications. Below the header, there's a summary bar with 'Code' (7 commits), 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Pulse', and 'Graphs'. On the right, there are buttons for 'Watch 1', 'Star 0', and 'Fork 0', with the 'Fork' button circled in blue. The main content area displays a commit history with three entries:

- betermieux changed minor detail (Latest commit 3dca929 34 seconds ago)
- .project added project (3 years ago)
- brief.txt changed minor detail (34 seconds ago)

At the bottom, there are links for GitHub terms like 'Contact GitHub', 'API', 'Training', 'Shop', 'Blog', and 'About'.

2,3,4: Änderungen durchführen

Die folgenden drei Schritte lokal durchführen:

- ▶ einen eigenen Branch vom master erstellen
- ▶ eigene Commits in diesen Branch erstellen
- ▶ alle Commits aus diesem Branch in das geforkte Projekt auf Github pushen

```
/ $ git clone https://github.com/karltoffel/GitProjekt.git
Cloning into 'GitProjekt'...
/ $ cd GitProjekt
/GitProjekt $ git checkout -b karls-zweig
Switched to a new branch 'karls-zweig'
/GitProjekt $ echo Orangen >> brief.txt
/GitProjekt $ git commit -a -m "Einkaufsliste ergänzt"
[karls-zweig 74d85ea] Einkaufsliste ergänzt
 1 file changed, 1 insertion(+)
/GitProjekt $ git push origin karls-zweig
Counting objects: 3, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/karltoffel/GitProjekt.git
 * [new branch]      karls-zweig -> karls-zweig
```

5: Pull-Request

Aus den Commits in einem Branch kann direkt ein Pull-Request beim Ursprungsprojekt erstellt werden:

The screenshot shows a GitHub repository page for 'karltoffel / GitProjekt'. The top navigation bar includes links for 'Pull requests', 'Issues', and 'Gist'. Below the repository name, it shows it was forked from 'betermieux/GitProjekt'. The main content area displays '7 commits', '3 branches', '1 release', and '1 contributor'. A section titled 'Your recently pushed branches:' lists 'karls-zweig (5 minutes ago)'. On the right side of this list is a green button labeled 'Compare & pull request', which is circled in blue. At the bottom of the page, there's a note about adding a README and a 'Add a README' button.

No description or website provided. — Edit

7 commits 3 branches 1 release 1 contributor

Your recently pushed branches:

karls-zweig (5 minutes ago) Compare & pull request

Branch: master New pull request Create new file Upload files Find file Clone or download

This branch is even with betermieux:master. Pull request Compare

betermieux changed minor detail Latest commit 3dca929 23 minutes ago

.project added project 3 years ago

brief.txt changed minor detail 23 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

5. Pull-Request

The screenshot shows a GitHub repository page for 'betermieux / GitProjekt'. The 'Code' tab is selected. At the top, there are buttons for 'Pull requests', 'Issues', and 'Gist'. On the right, there are buttons for 'Watch' (1), 'Star' (0), and 'Fork' (1). Below the navigation, there are links for 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Pulse', and 'Graphs'. A large heading 'Open a pull request' is displayed, followed by the instruction 'Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.' Below this, there are dropdown menus for 'base fork: bettermieux/GitProjekt', 'base: master', '...', 'head fork: karltoffel/GitProjekt', and 'compare: karls-zweig'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' A note from a collaborator, 'Einkaufsliste ergänzt', is shown in a rich text editor window. The note reads: 'Ich habe einen Eintrag auf der Einkaufsliste hinzugefügt. Bitte im Hauptprojekt übernehmen.' Below the note, there is a placeholder for attachments: 'Attach files by dragging & dropping, selecting them, or pasting from the clipboard.' A checkbox 'Allow edits from maintainers.' is checked, with a link 'Learn more' next to it. A large green button at the bottom right is labeled 'Create pull request', which is circled in blue.

6: Diskussionen führen

Der Besitzer des Ursprungsprojekts bekommt eine Benachrichtigung und sieht den Pull-Request in einer Liste:

The screenshot shows a GitHub repository page for 'betermieux / GitProjekt'. The 'Pull requests' tab is selected, showing one open pull request. A hand-drawn blue oval highlights this pull request. The pull request details are as follows:

- Title: Einkaufsliste ergänzt
- Description: #1 opened 3 minutes ago by karltoffel

7: Branch mergen

The screenshot shows a GitHub repository page for 'betermieux / GitProjekt'. The 'Pull requests' tab is selected, showing one open pull request titled 'Einkaufsliste ergänzt #1'. The pull request details show a commit from 'karltoffel' merging into 'betermieux:master' from 'karltoffel:karls-zweig'. A comment from 'karltoffel' states: 'Ich habe einen Eintrag auf der Einkaufsliste hinzugefügt. Bitte im Hauptprojekt übernehmen.' Below the comment, there are sections for 'Conversation' (0), 'Commits' (1), and 'Files changed' (1). The commit message is 'Einkaufsliste ergänzt' with hash '74d85ea'. A review button 'Add your review' is present. At the bottom, a green button 'Merge pull request' is highlighted with a blue oval. To the right, there are project, label, milestone, assignee, and participant details. The URL of the page is <https://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project>.

einfache Textgestaltung

Exkurs: Markdown

Markdown

- Auszeichnungssprache zur schnellen Formatierung von Text
 - ▶ fett, kursiv, Listen, Überschriften
- Verwendet nur gebräuchliche Sonderzeichen
- Vergleichbar mit »Wikitext« von »Wikipedia«
- Wir intern in HTML-Quelltext umgewandelt
 - ▶ kann aber auch in PDF oder LaTeX umgewandelt werden
- Wird von vielen Entwicklern verwendet:
 - ▶ Stack Overflow
 - ▶ Google+
 - ▶ GitHub

Markdown Syntax

Ausgangsform	Zielform
Normaler Text wird so dargestellt wie eingegeben. Eine Leerzeile erzeugt einen Absatz	Normaler Text wird so dargestellt wie eingegeben. Eine Leerzeile erzeugt einen Absatz.
Kursiv, **Fett** und ***Fett kursiv*** bzw. <u>Kursiv</u> , <u>Fett</u> und <u>Fett kursiv</u>	<i>Kursiv</i> , Fett und Fett kursiv bzw. <i>Kursiv</i> , Fett und Fett kursiv
Markiert Text als `Quelltext`	Markiert Text als Quelltext
* Ein Punkt in einer ungeordneten Liste * Ein weiterer Punkt in einer ungeordneten Liste * Ein Unterpunkt, um vier Leerzeichen eingerückt * Statt * funktionieren auch + oder -	<ul style="list-style-type: none">• Ein Punkt in einer ungeordneten Liste• Ein weiterer Punkt in einer ungeordneten Liste<ul style="list-style-type: none">• Ein Unterpunkt, um vier Leerzeichen eingerückt• Statt * funktionieren auch + oder -

Markdown Syntax

Ausgangsform	Zielform
# Überschrift in Ebene 1 #### Überschrift in Ebene 4	Überschrift in Ebene 1 Überschrift in Ebene 4
> Dieses Zitat wird in ein HTML-Blockquote-Element gepackt.	Dieses Zitat wird in ein HTML-Blockquote-Element gepackt.
[Beschriftung des Hyperlinks](http://de.wikipedia.org/"Titel, der beim Überfahren mit der Maus angezeigt wird")	<u>Beschriftung des Hyperlinks</u>
![Alternativtext](Bild-URL "Bildtitel hier")	Bindet ein Bild von der Quelle Bild-URL ein.

Markdown Beispiel

```
# A First Level Header
```

```
## A Second Level Header
```

```
Now is the time for all good men  
to come to the aid of their  
country. This is just a regular  
paragraph.
```

```
The quick brown fox jumped over  
the lazy dog's back.
```

```
### Header 3
```

```
> This is a blockquote.
```

```
>
```

```
> This is the second paragraph in the blockquote.
```

```
>
```

```
> ## This is an H2 in a blockquote
```

```
Some of these words *are emphasized*.
```

```
Some of these words _are emphasized also_.
```

```
Use two asterisks for **strong emphasis**.
```

```
Or, if you prefer, __use two underscores instead__.
```

A First Level Header

A Second Level Header

Now is the time for all good men to come to the aid of their country. This is just a regular paragraph.

The quick brown fox jumped over the lazy dog's back.

Header 3

This is a blockquote.

This is the second paragraph in the blockquote.

This is an H2 in a blockquote

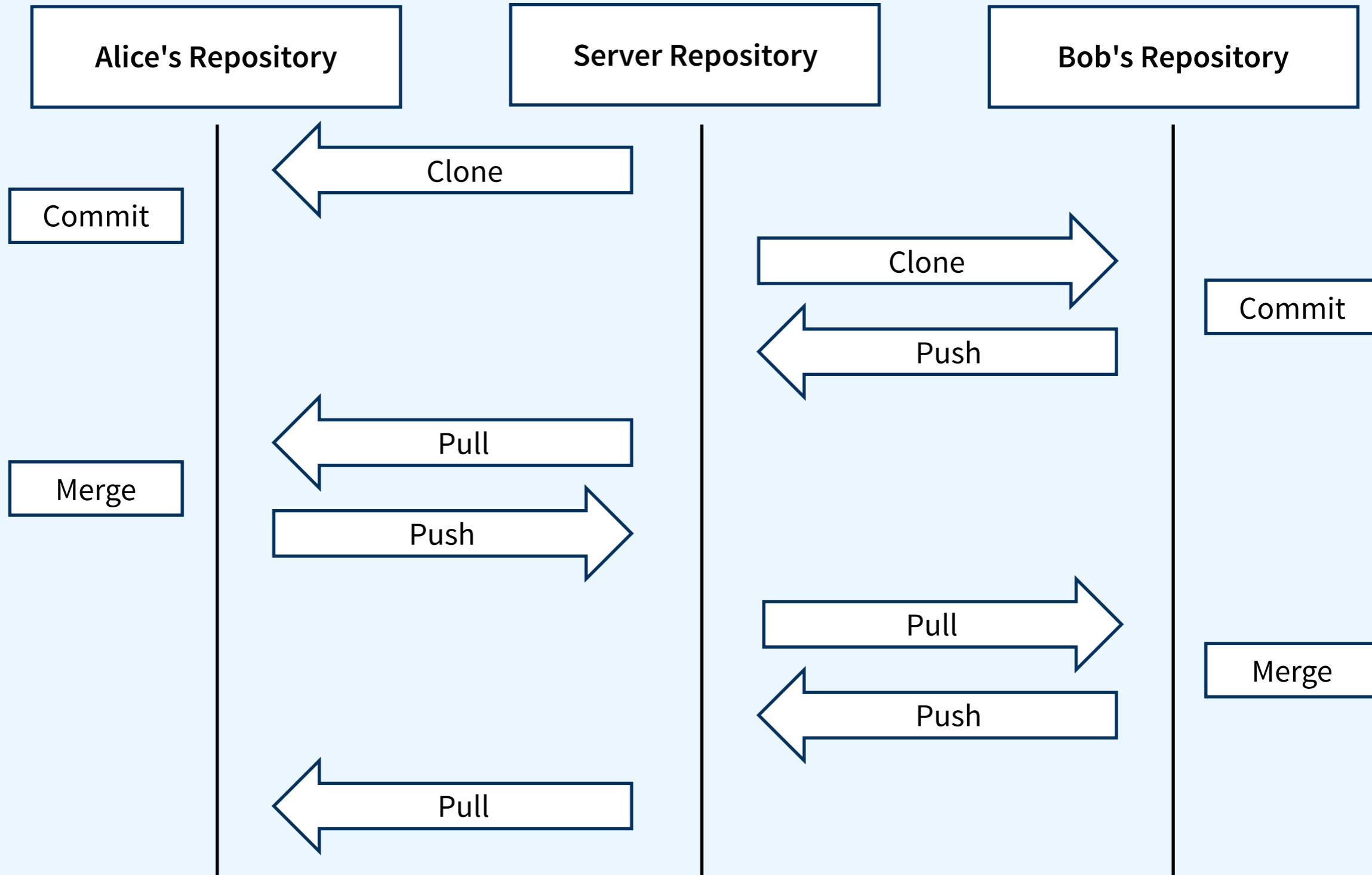
Some of these words *are emphasized*. Some of these words *are emphasized also*.

Use two asterisks for **strong emphasis**. Or, if you prefer, **use two underscores instead**.

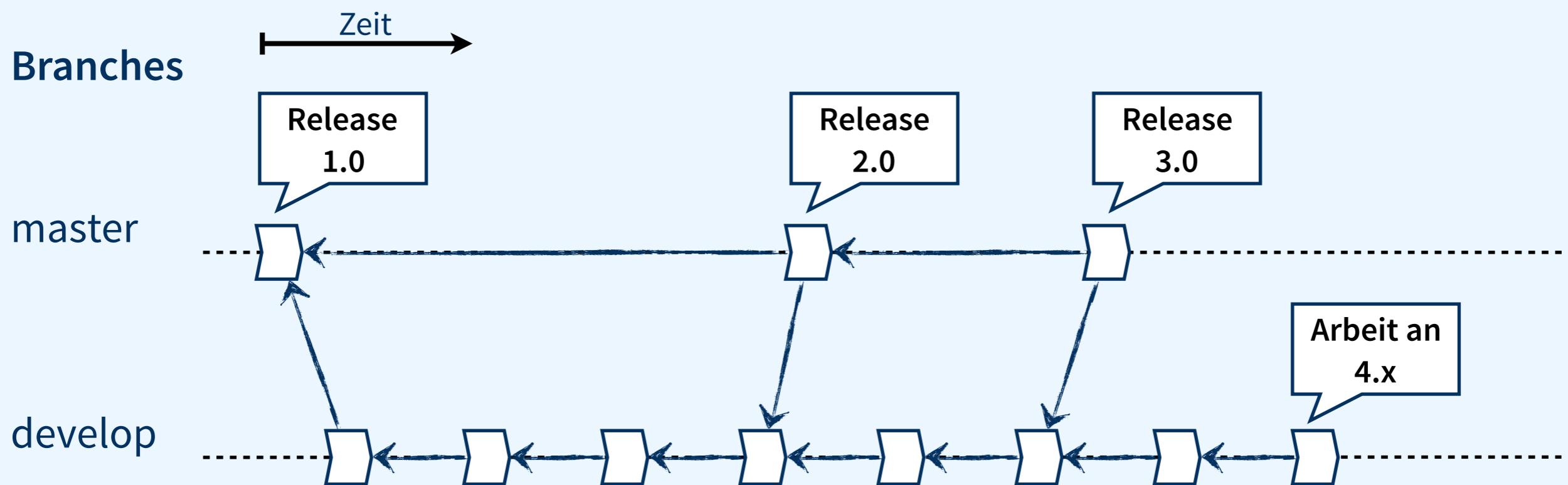
Arbeitsabläufe bei der Entwicklung

Git Workflows

Einfacher Workflow

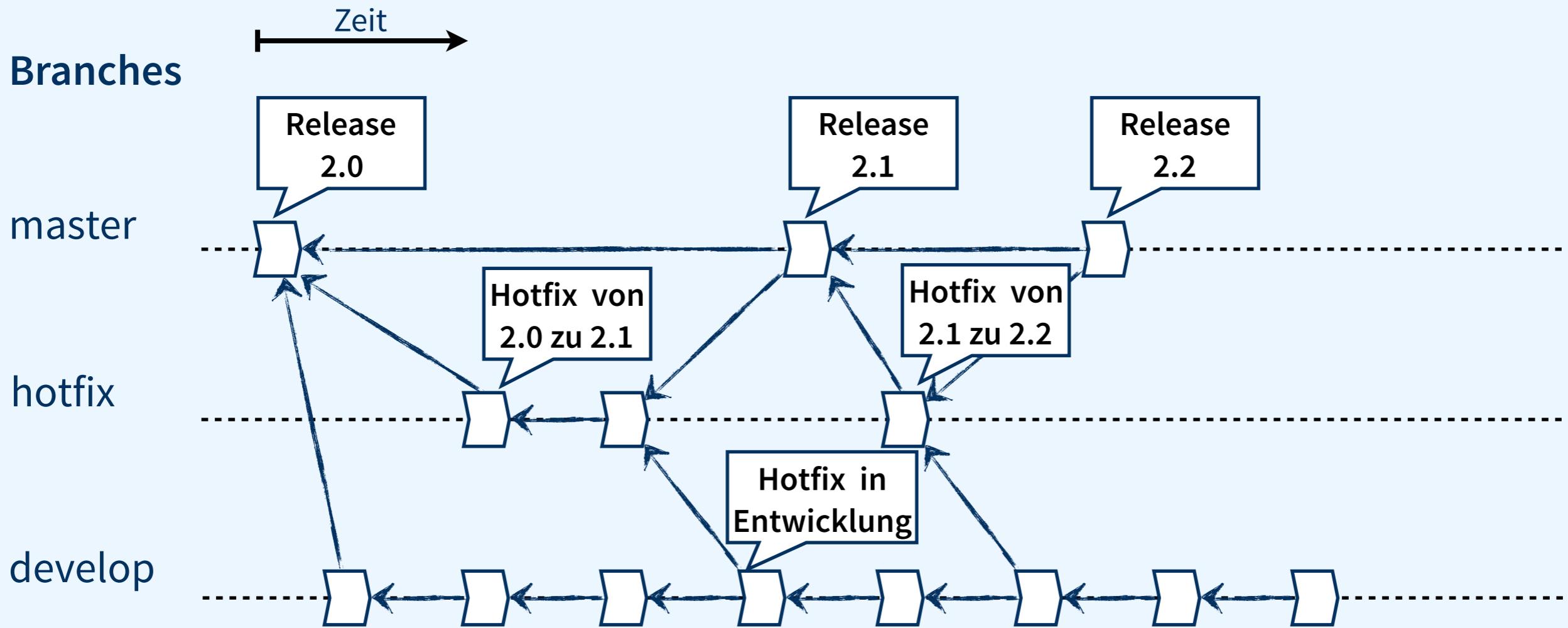


Workflow mit Branches



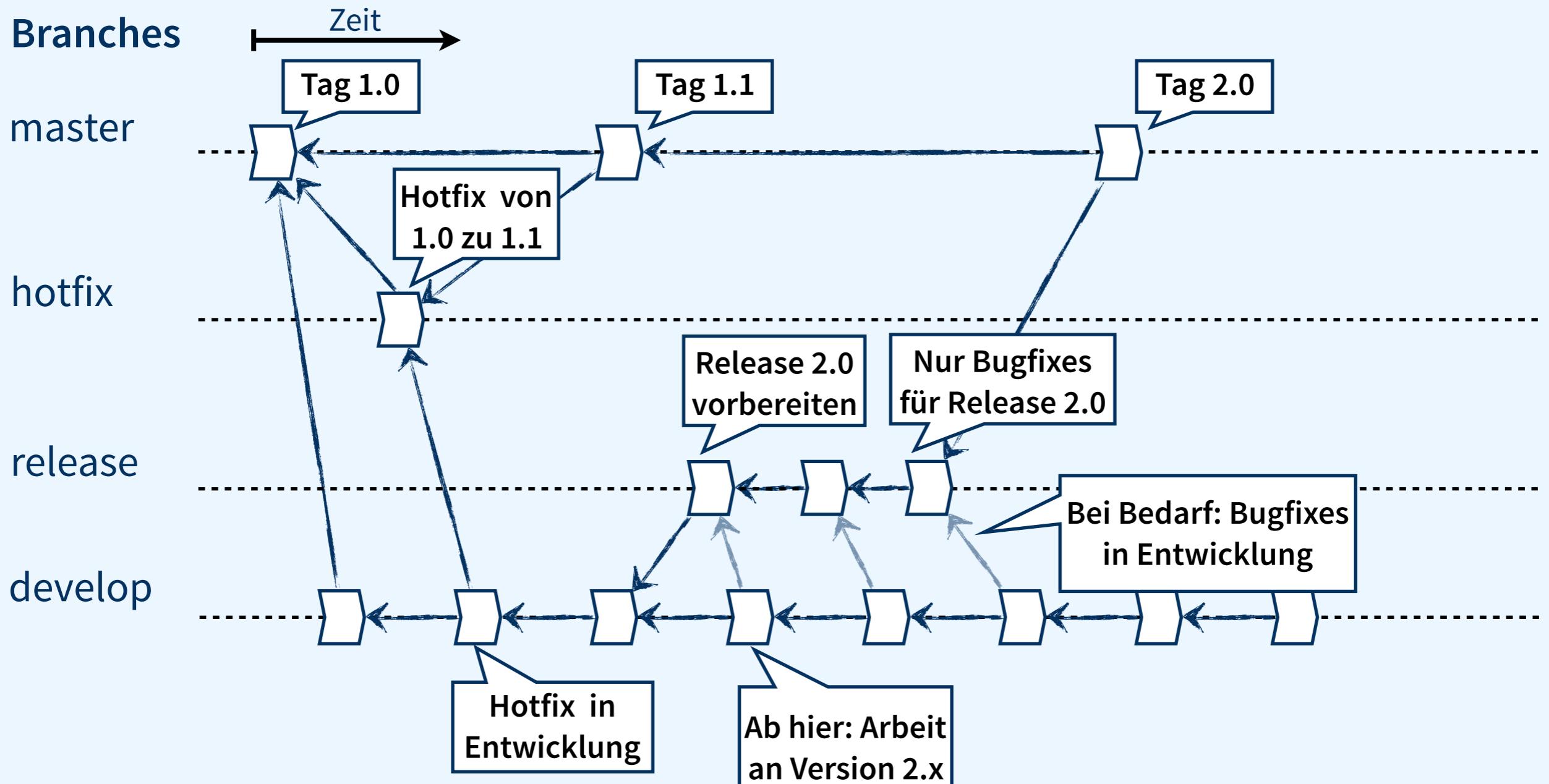
- Master-Branch enthält nur veröffentlichte Releases
- Develop-Branch enthält aktuelle Änderungen aller Entwickler (Integrationszweig)

Hotfix Branches



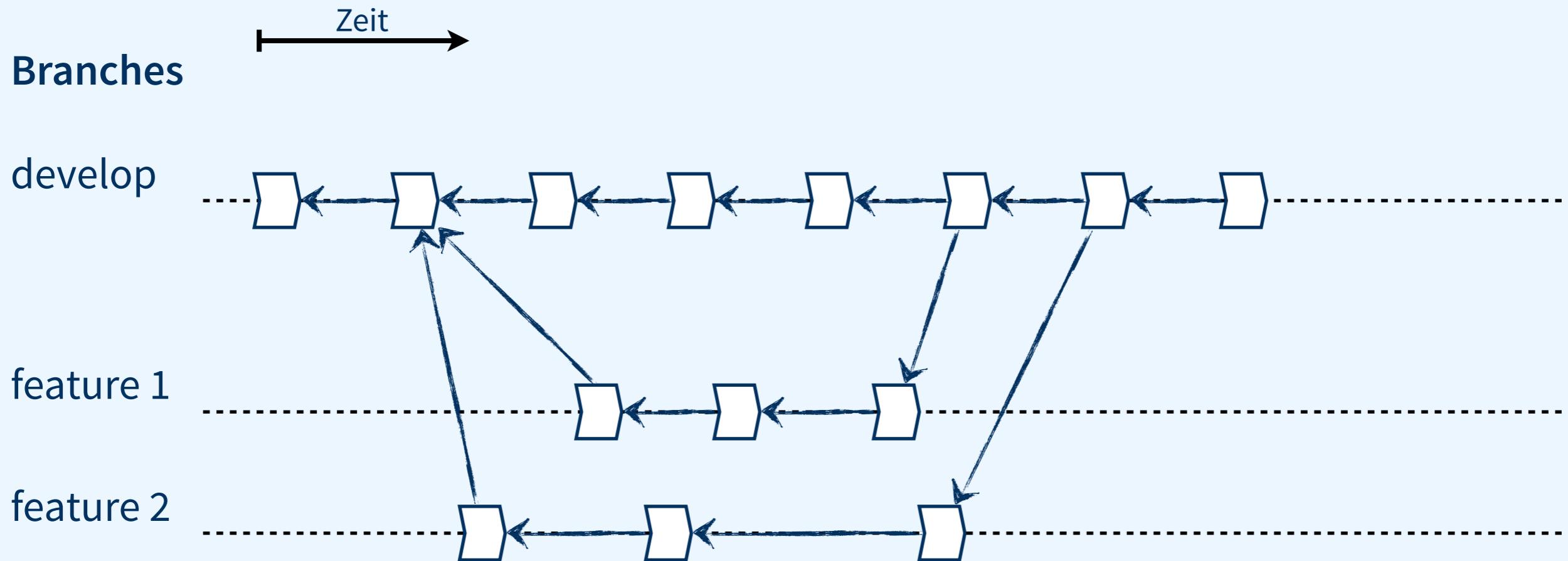
- Hotfixes korrigieren schwerwiegende Fehler (Sicherheitslücken, ...)
- Hotfixes führen zu einem neuen Release
- Hotfixes sollten auch in den Entwicklungszweig übernommen werden

Release Branches



Release-Branches erlauben eine neue Veröffentlichung zu stabilisieren
(Alpha-Version, Beta-Version, Release Candidates, ...)

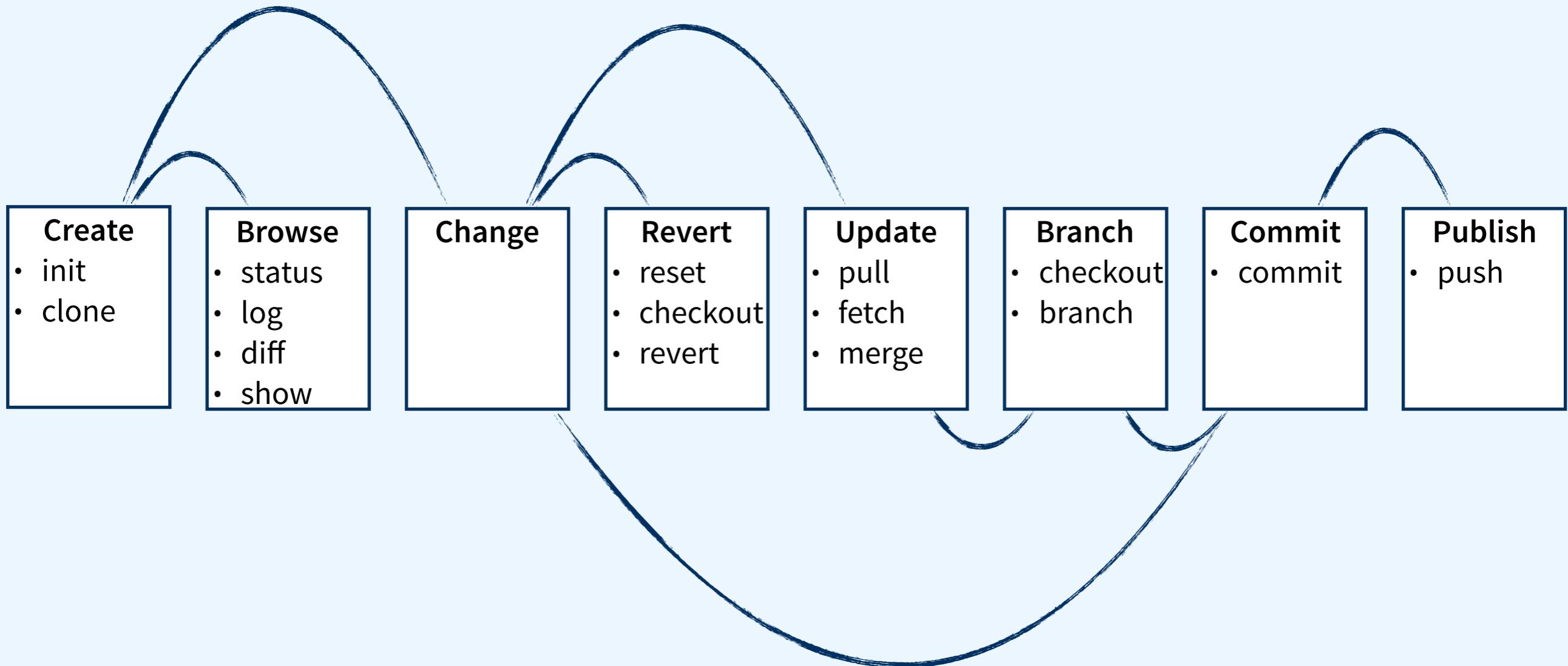
Feature Branches



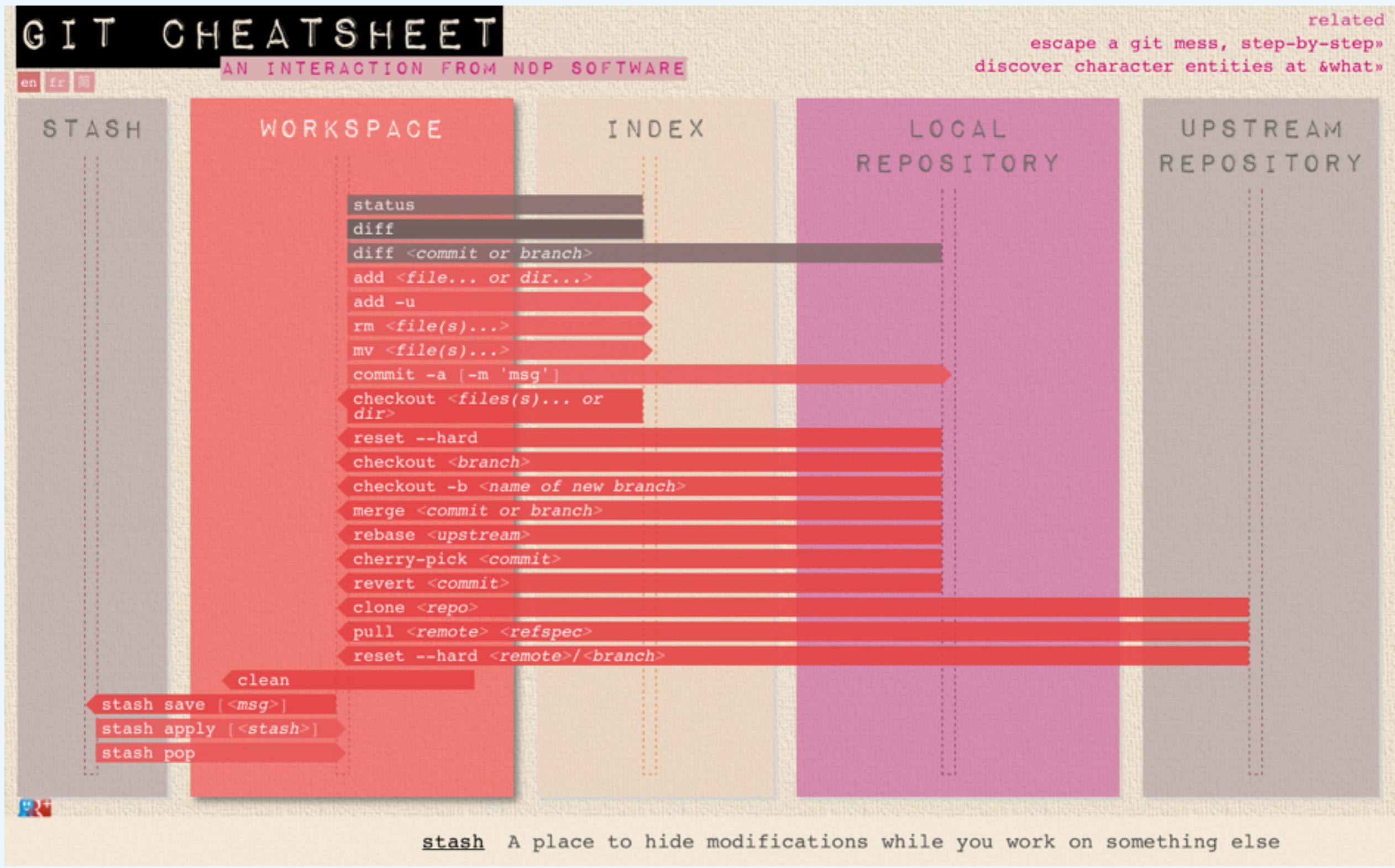
- Feature-Branches müssen vom Develop-Branch verzweigen und wieder mit ihm vereinigt werden
- Es können mehrere Feature-Branches parallel existieren

Sequenz

Üblicher Arbeitsablauf bei der Verwendung von Git:

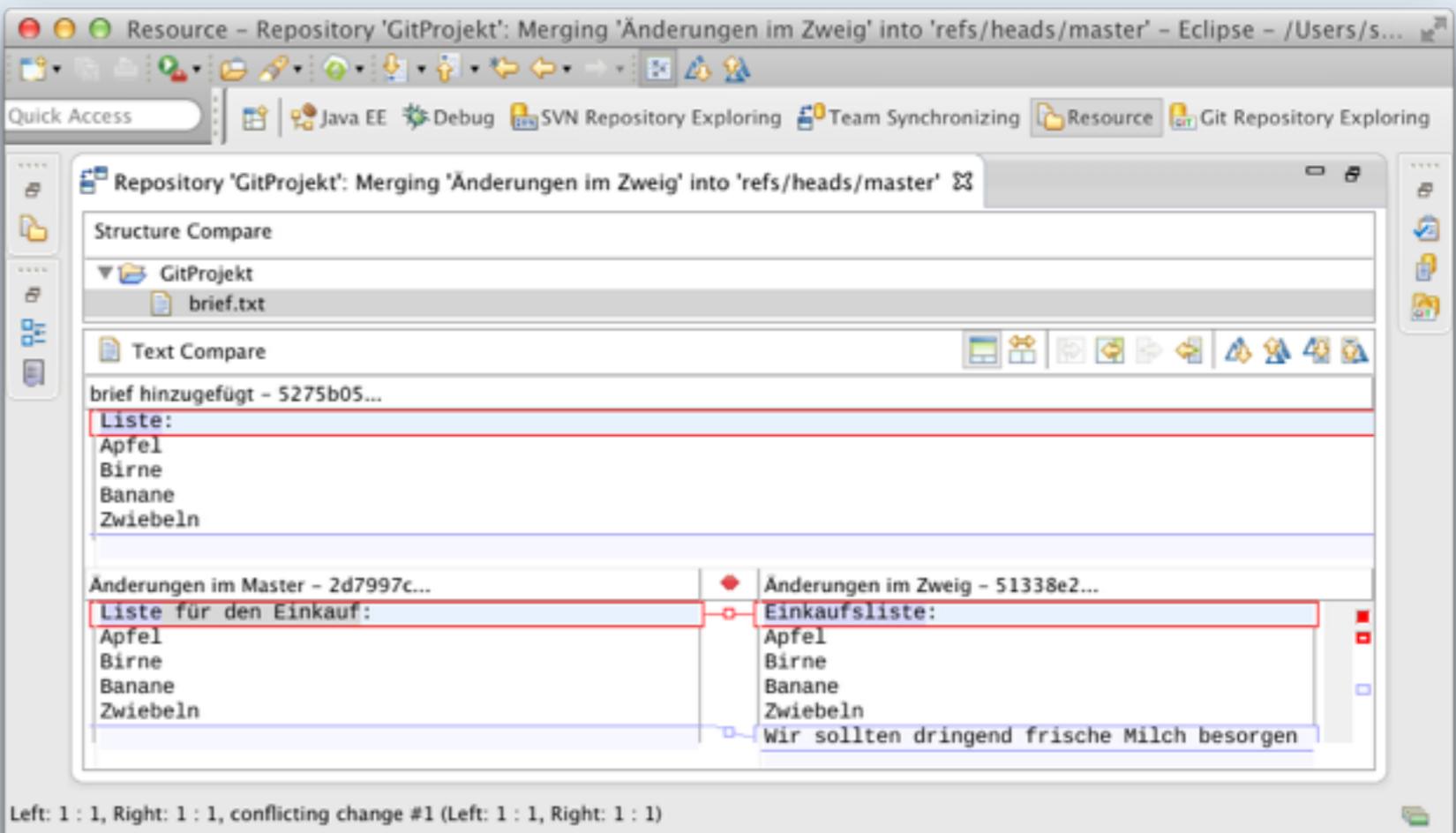


Interactive Cheatsheet

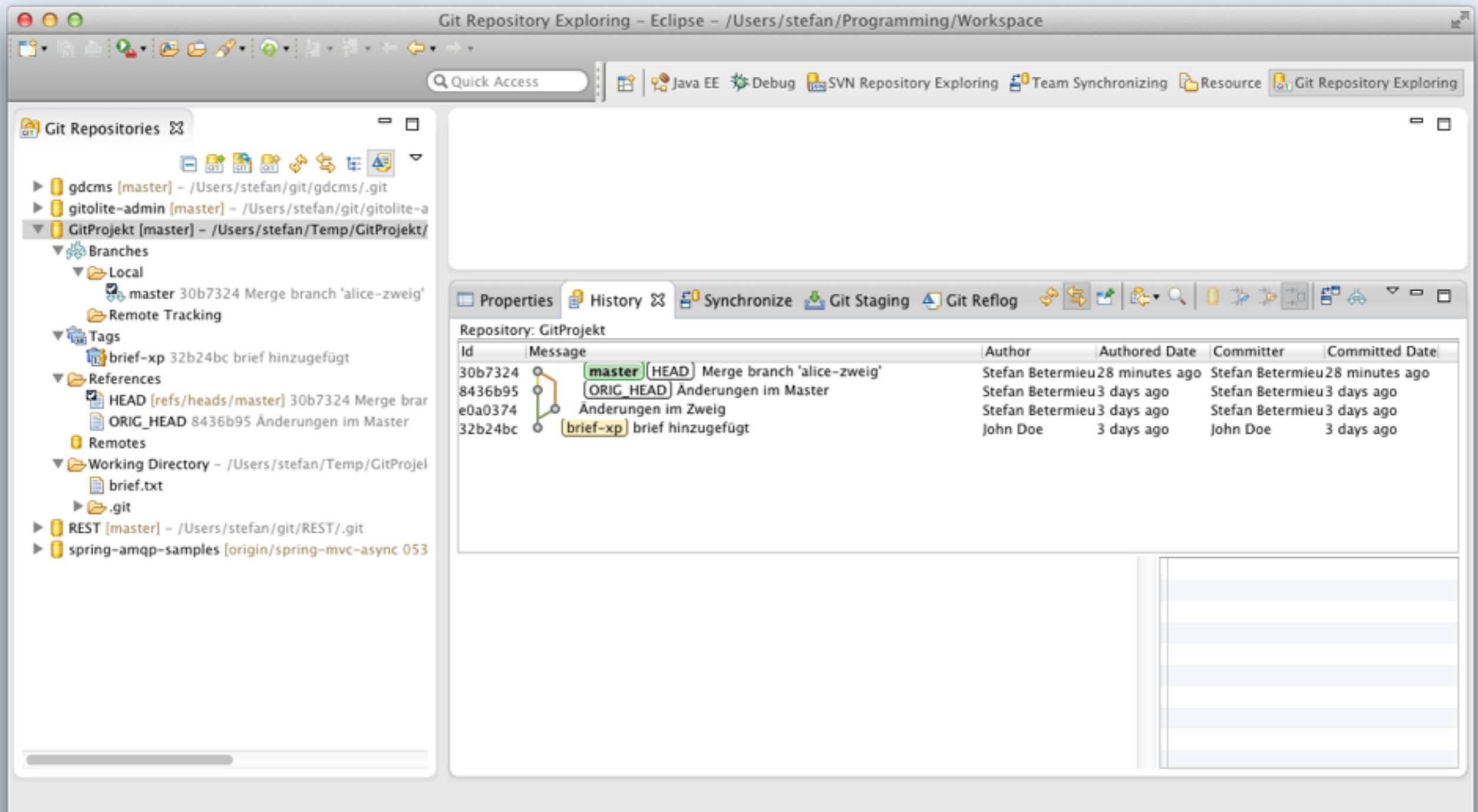


GIT Plugin für Eclipse

- Eclipse bietet in der JavaEE Version ein vollständiges Git-Plugin:
 - ▶ <http://eclipse.org/egit/>
 - ▶ benötigt keine weiteren Plugins
 - ▶ kann in anderen Eclipse-Varianten auch nachinstalliert werden
- Konflikte lösen:
»three-way-merge«
 - ▶ Zielzweig
 - ▶ Quellzweig
 - ▶ gemeinsamer Vorfahr



Demo Eclipse EGIT





ZUSAMMENFASSUNG

Best Practices

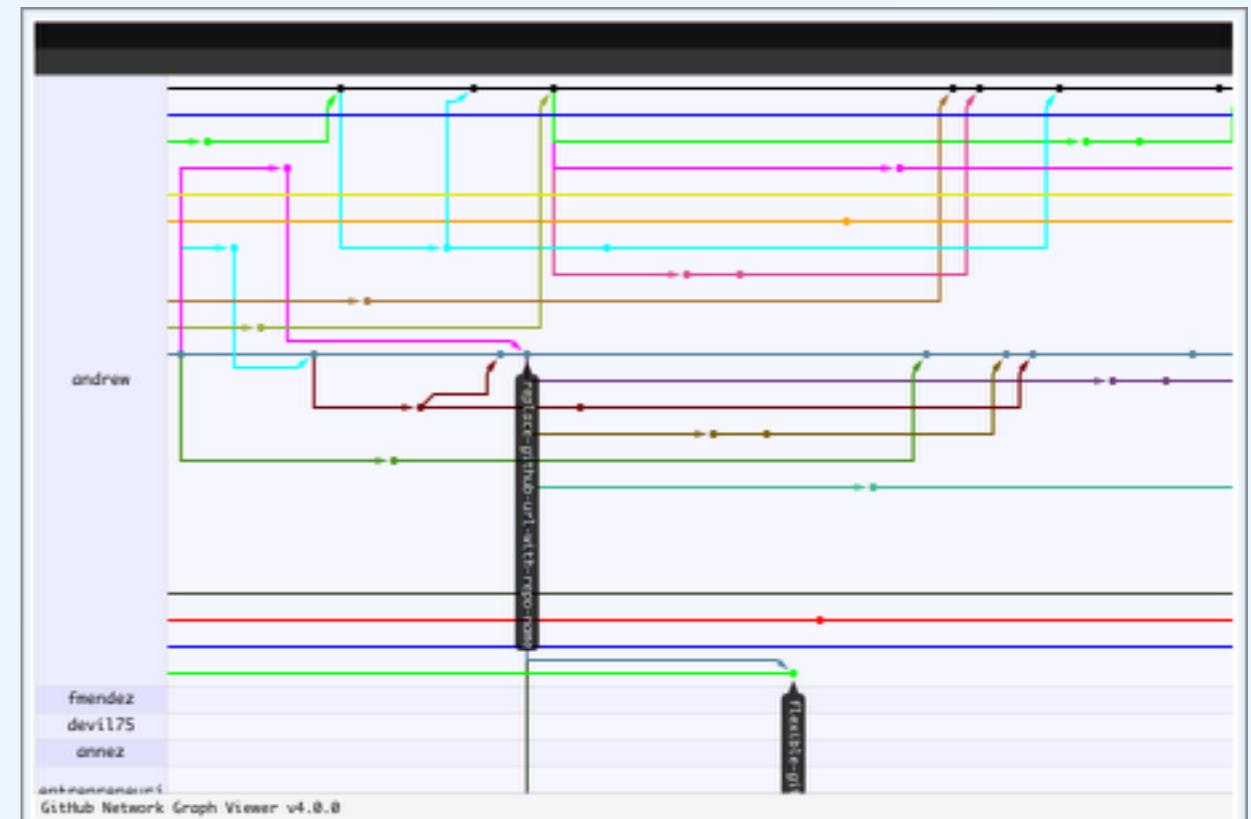
- Thematisch passend committen
 - ▶ zwei Bugs in dergleichen Datei nicht in einem Commit
 - ▶ Staging Area verwenden
- Häufig committen
 - ▶ Team profitiert schnell von Änderungen
 - ▶ weniger Merge-Probleme
- Nichts halbfertiges comitten
 - ▶ große Änderungen in kleinere logische Änderungen teilen
 - ▶ kein Feierabend-Commit

Best Practices

- Testen vor dem committen
 - ▶ Code sollte kompilierbar sein
 - ▶ Tests sollten erstellt und durchlaufen werden
- Gute Commit-Messages
 - ▶ Erste Zeile enthält eine Zusammenfassung (<50 Zeichen)
 - ▶ Danach »warum« und »wie« der Änderung
- Versionskontrolle ist kein Backup
 - ▶ siehe »Thematisch passend committen«
- Workflow im Team absprechen
 - ▶ wofür Branches, wie granular die Commits, etc...

Gefahren

- Gefahr des Datenverlustes, wenn Nutzer lange auf dem lokalen Repository arbeiten
 - ▶ Server lassen sich leichter sichern als Clients
 - ▶ lokales Repository häufig mit Server mergen
- Gefahr der auseinanderlaufenden Branches, wenn Nutzer lange auf dem lokalen Repository arbeiten
 - ▶ »branching hell«
 - ▶ lokales Repository häufig mit Server mergen



DANKE