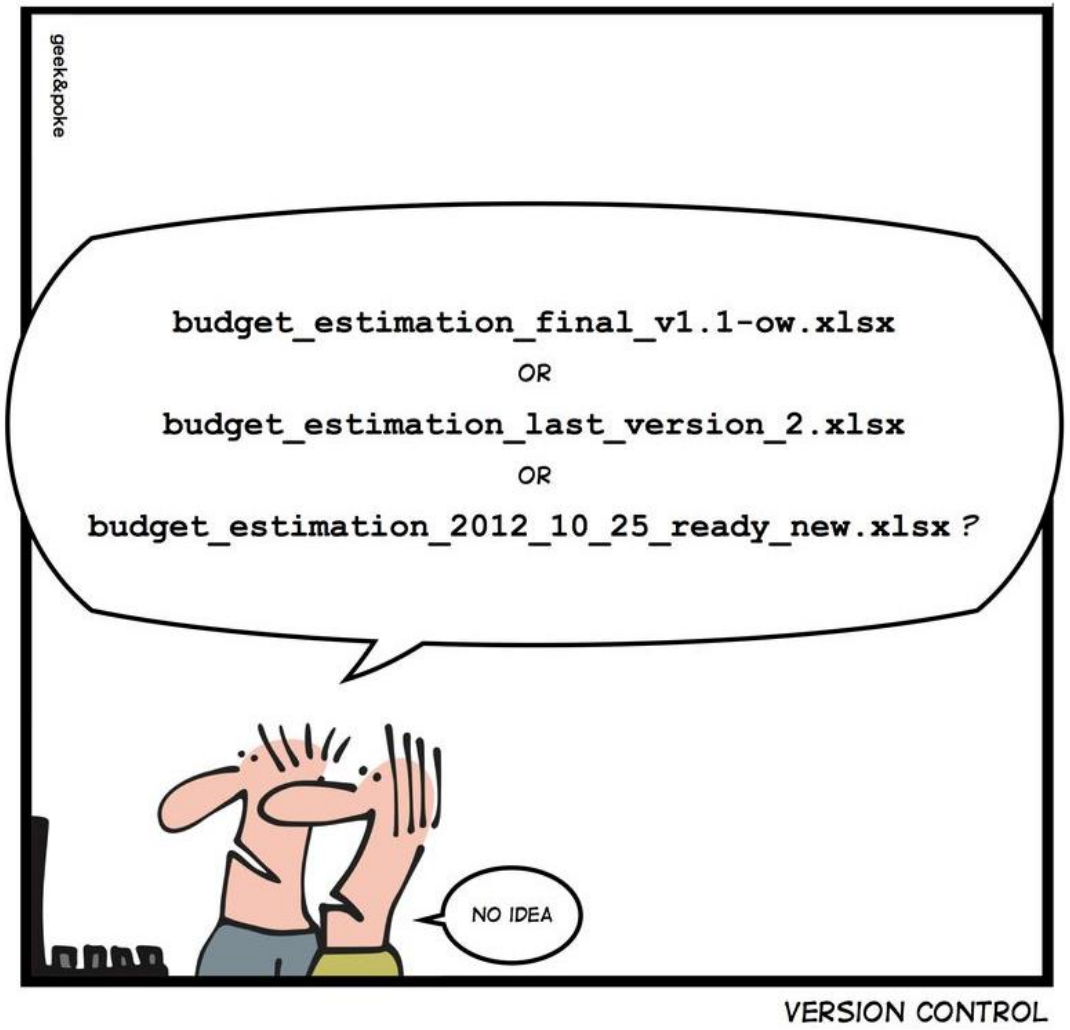
Versionsverwaltung / Git

HAW Hamburg / Fachbereich Informatik

Tim Lüecke

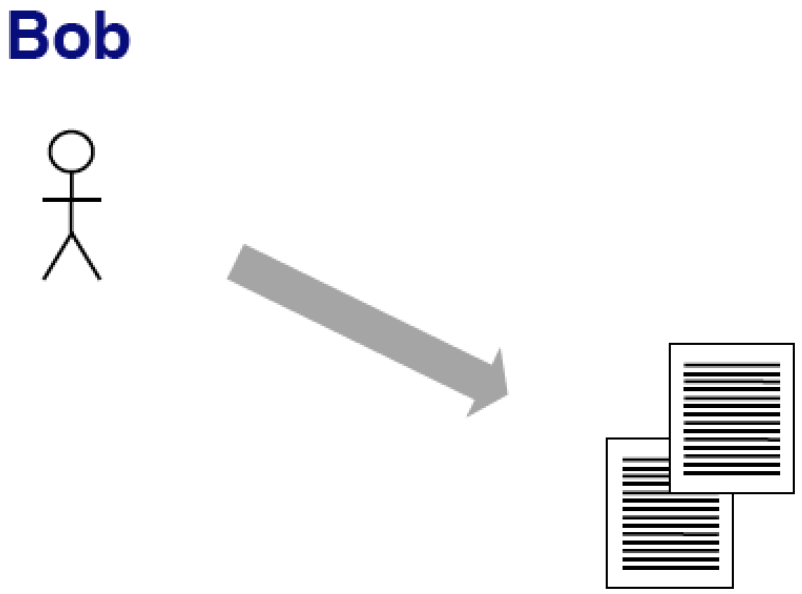
(Tim.Lueecke@haw-hamburg.de)



Agenda

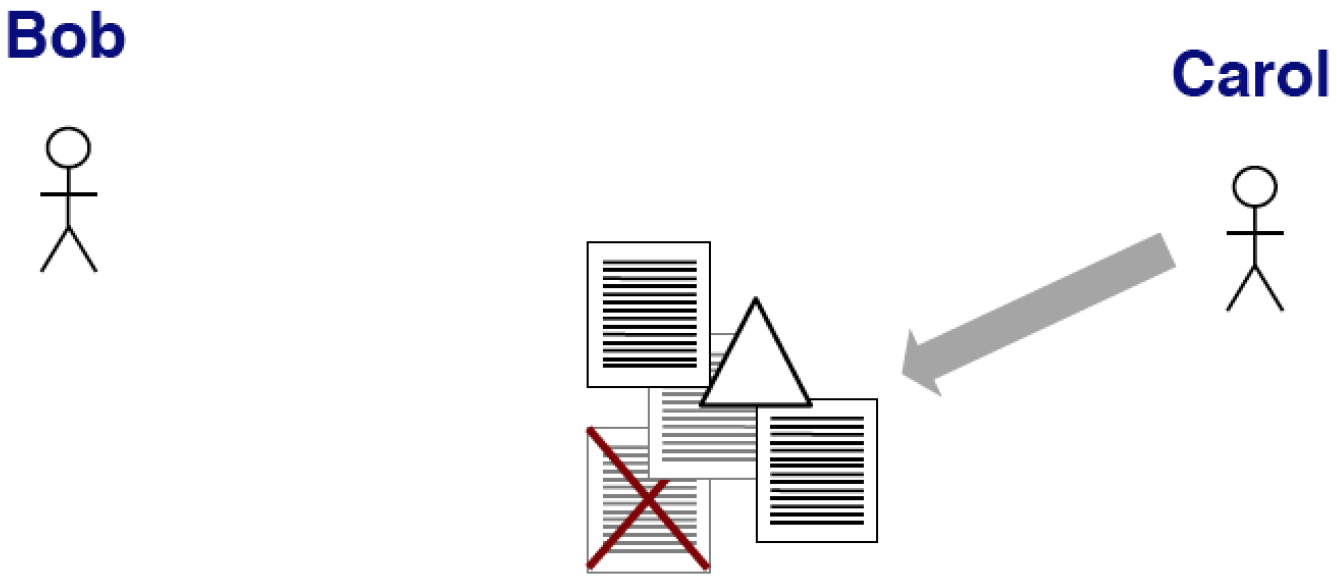
- **Motivation**
- Zentralisierter Ansatz
- Dezentralisierter Ansatz mit Git
- Workflow Modelle
- Zusammenfassung

Motivation



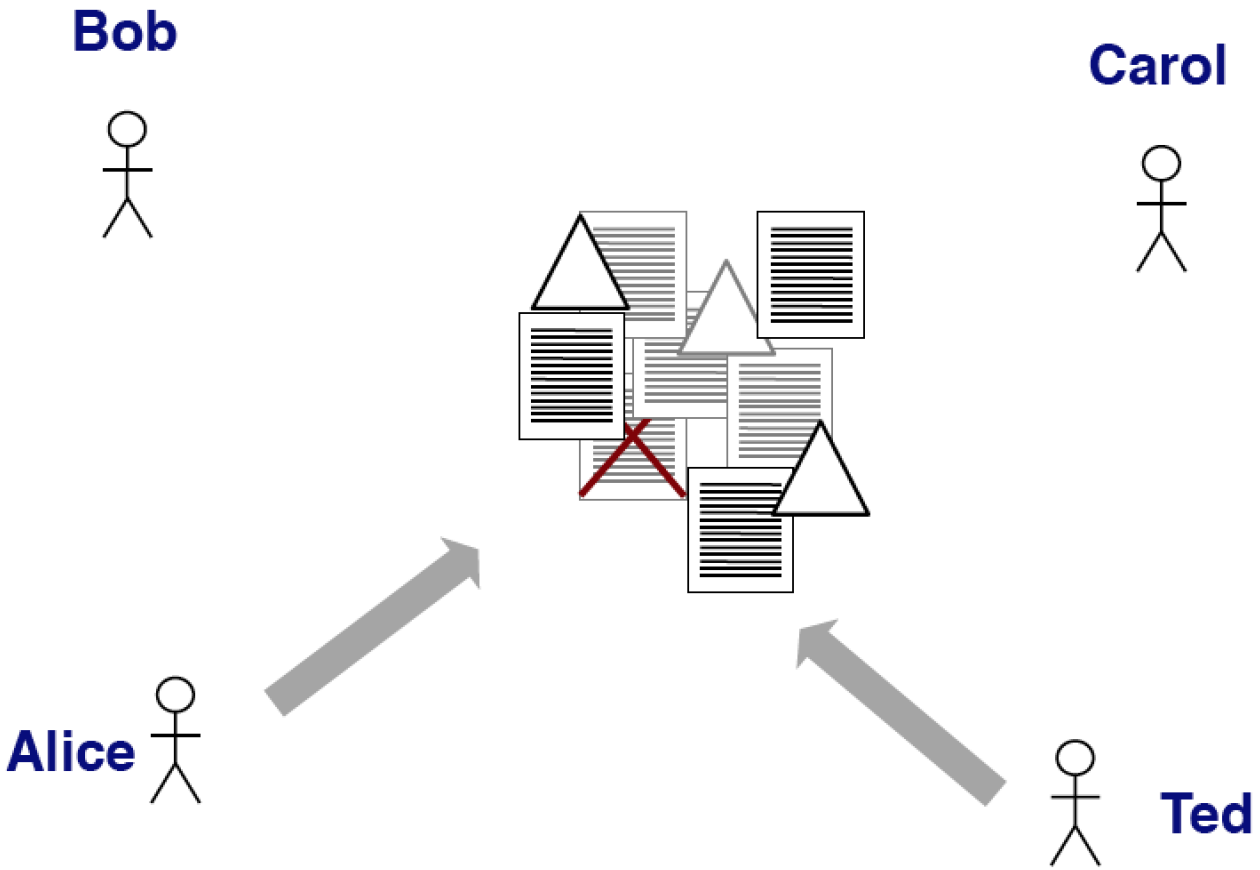
Quelle: http://de.slideshare.net/jomikr/quick-introduction-to-git

Motivation



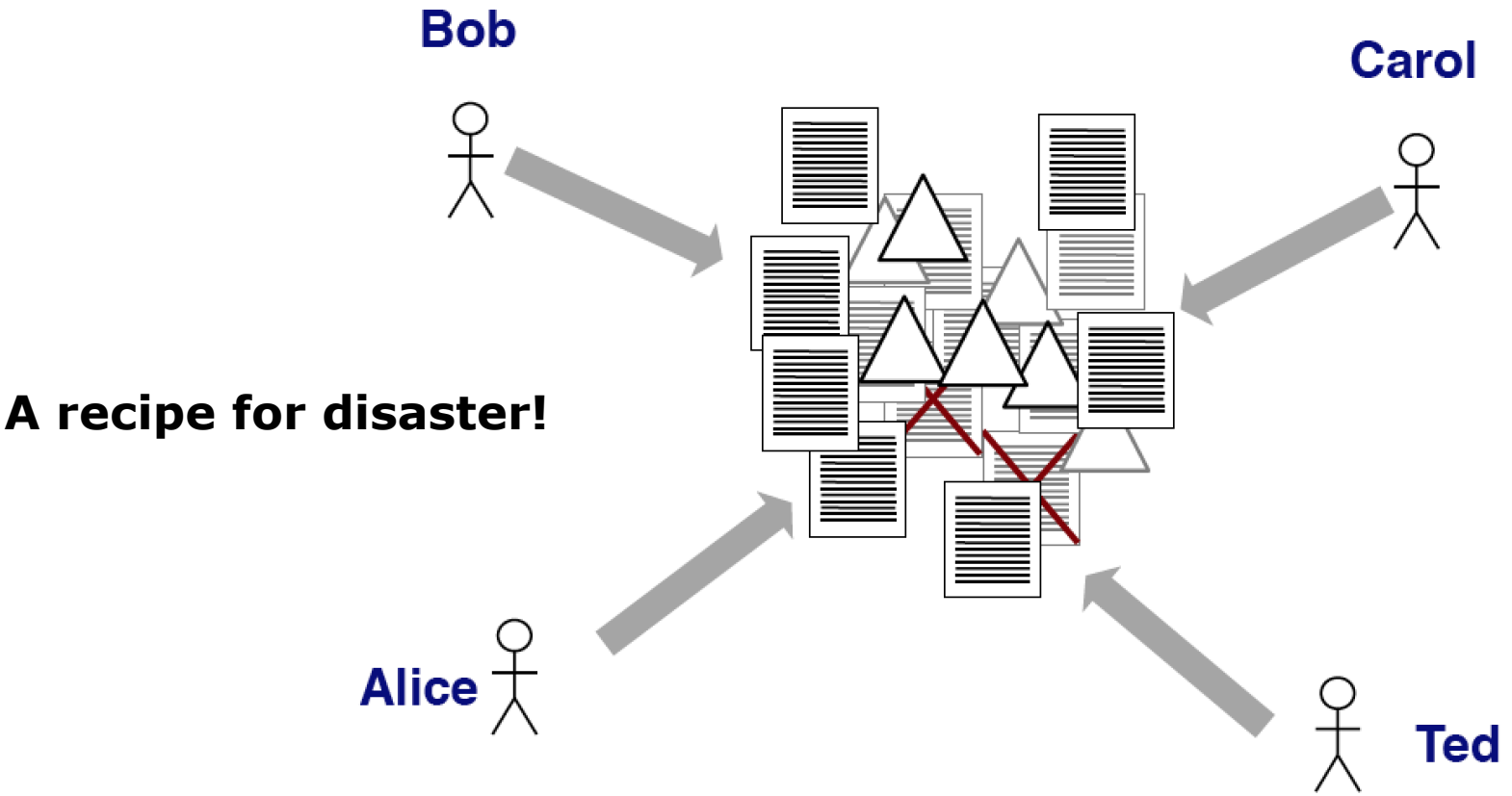
Quelle: http://de.slideshare.net/jomikr/quick-introduction-to-git

Motivation



Quelle: http://de.slideshare.net/jomikr/quick-introduction-to-git

Motivation

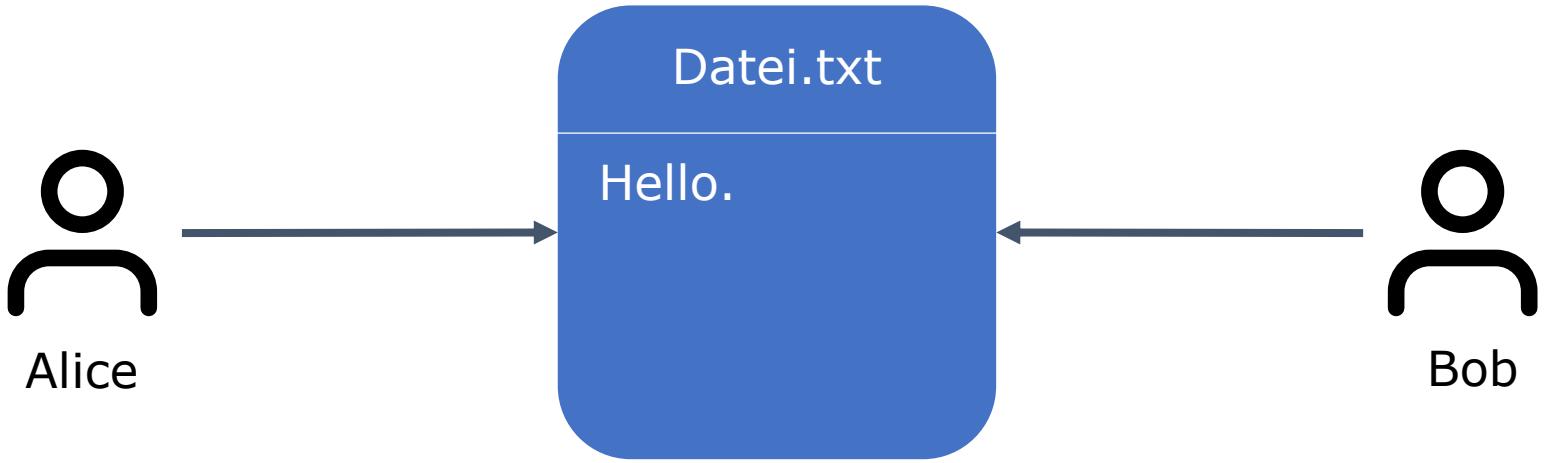


Quelle: http://de.slideshare.net/jomikr/quick-introduction-to-git

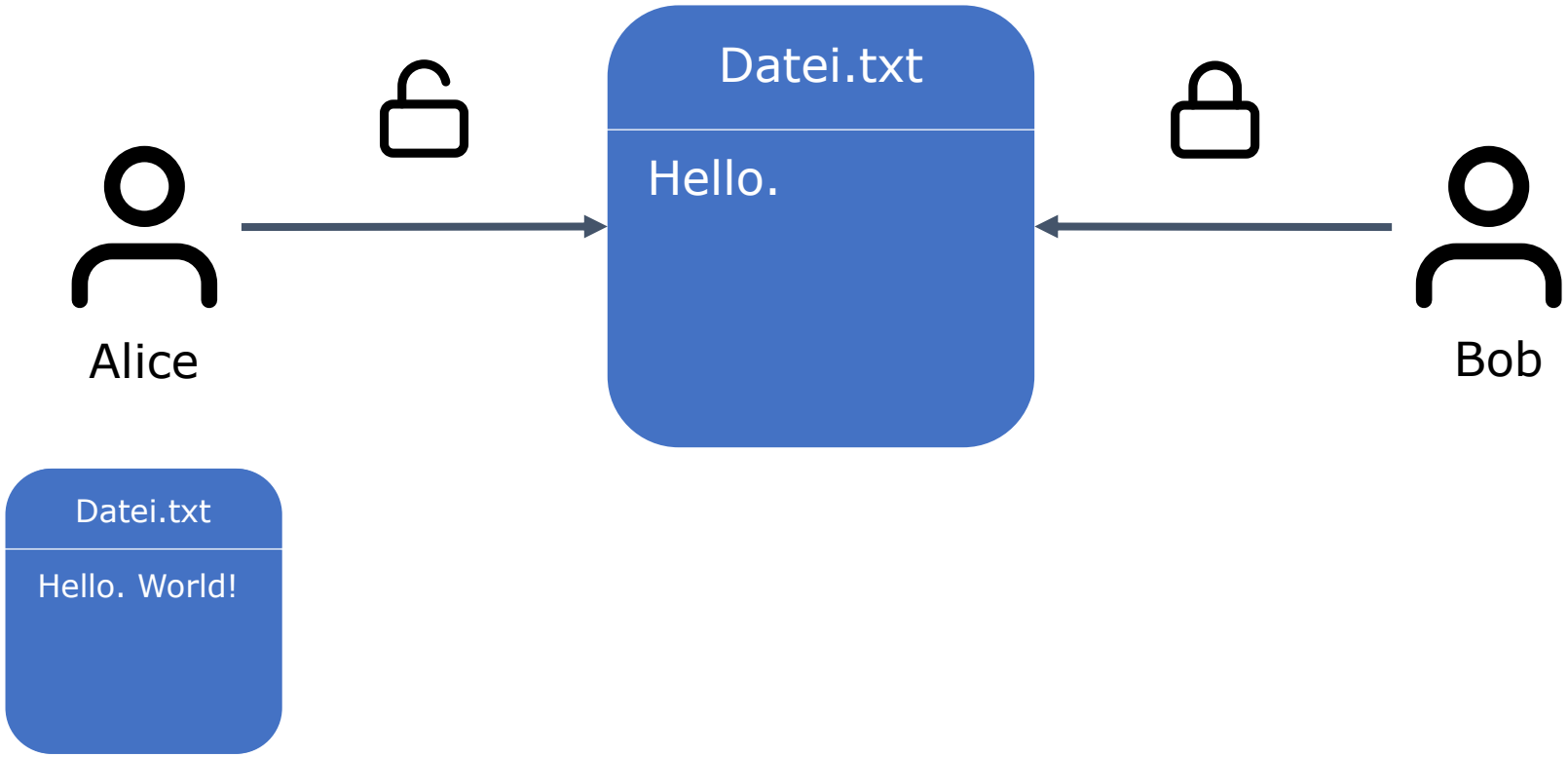
Agenda

- Motivation
- **Zentralisierter Ansatz**
- Dezentralisierter Ansatz mit Git
- Workflow Modelle
- Zusammenfassung

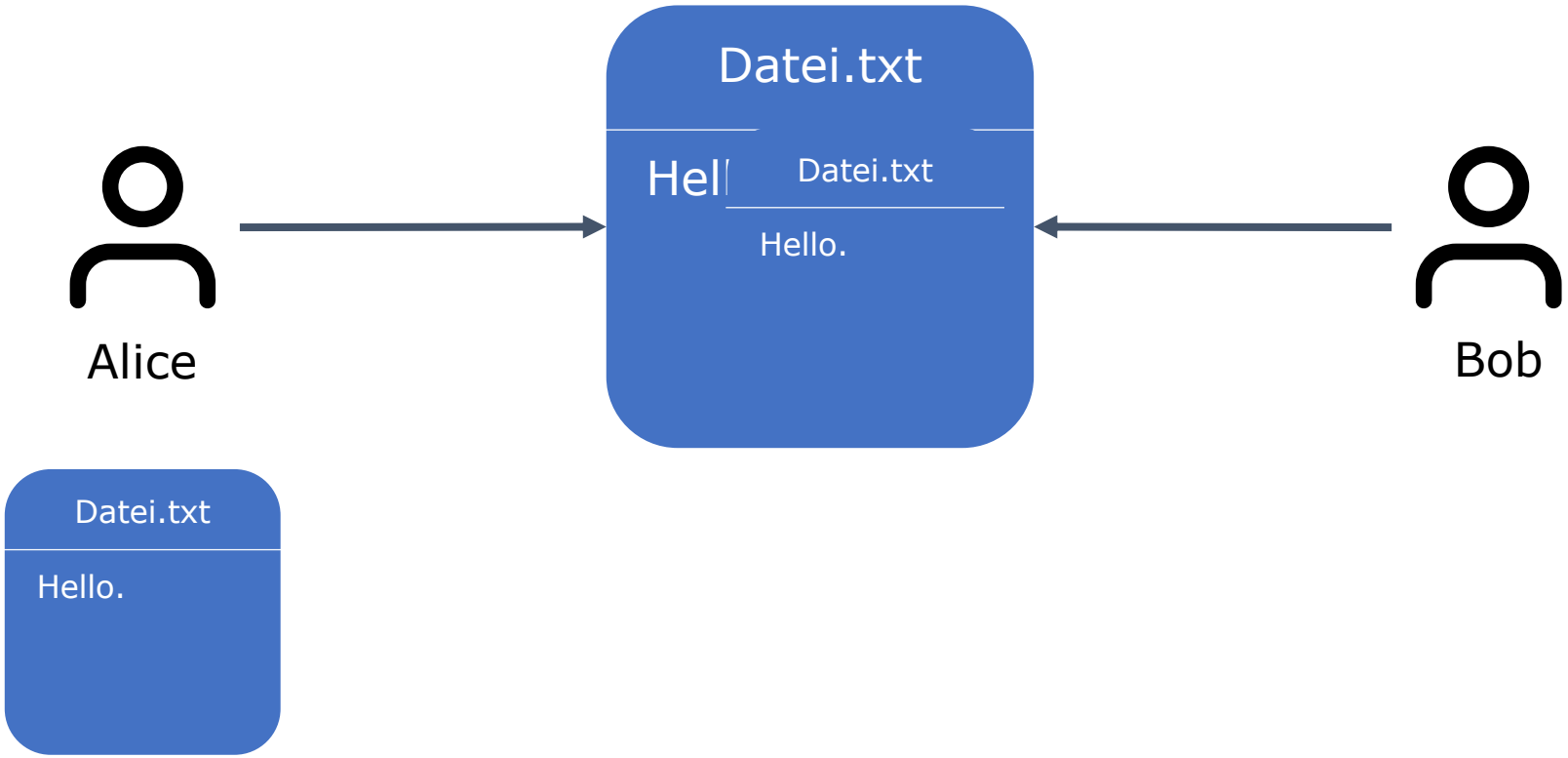
Eine Datei, zwei Bearbeiter



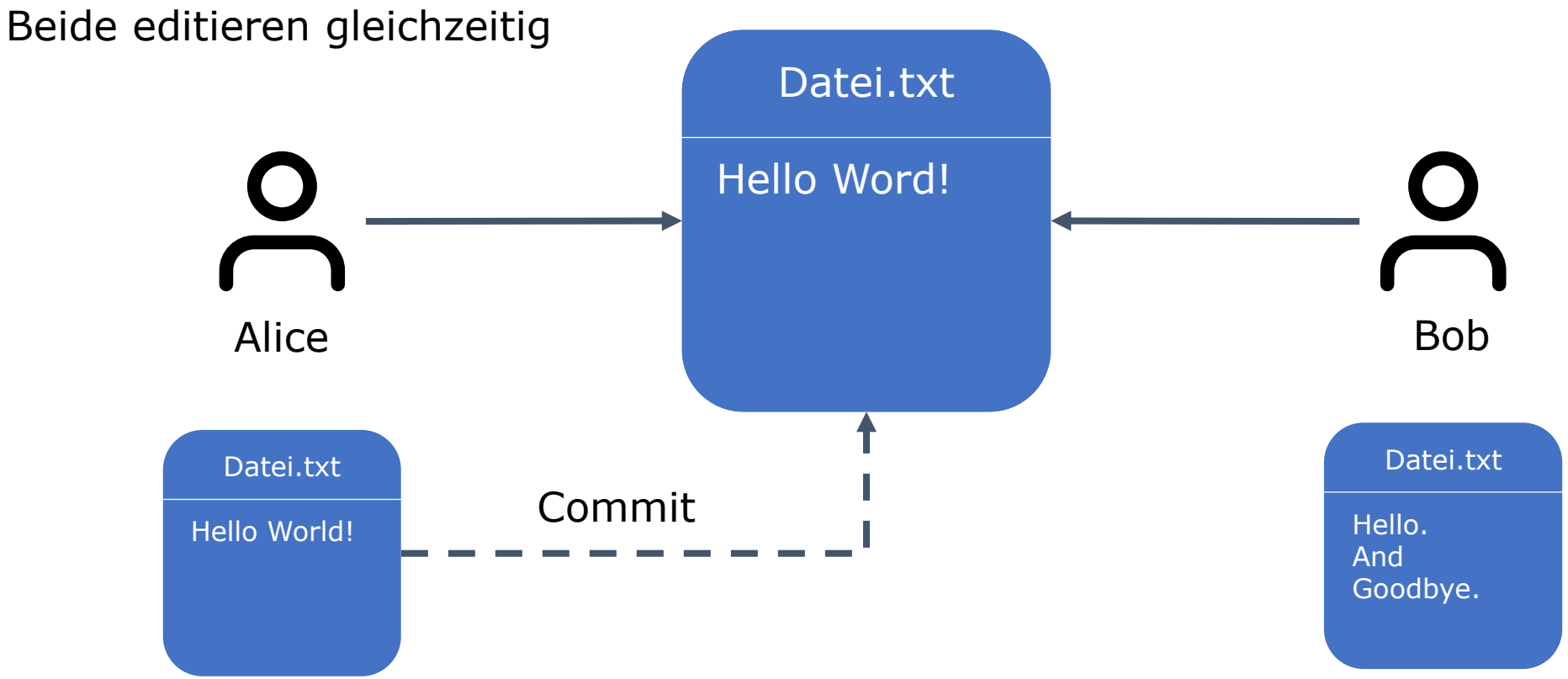
Lineare Vorgehensweise: Checkout + Edit



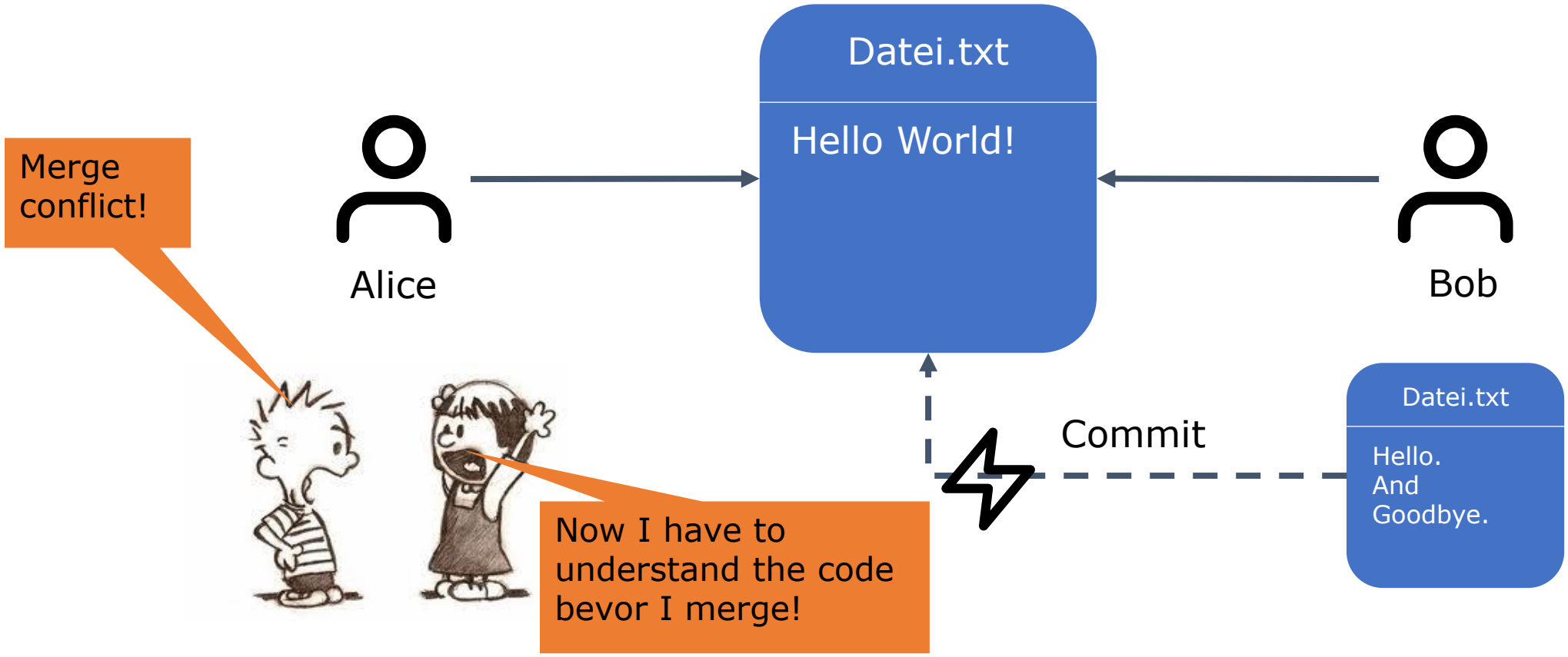
Nicht-linearer Ansatz: beide holen sich eine Version



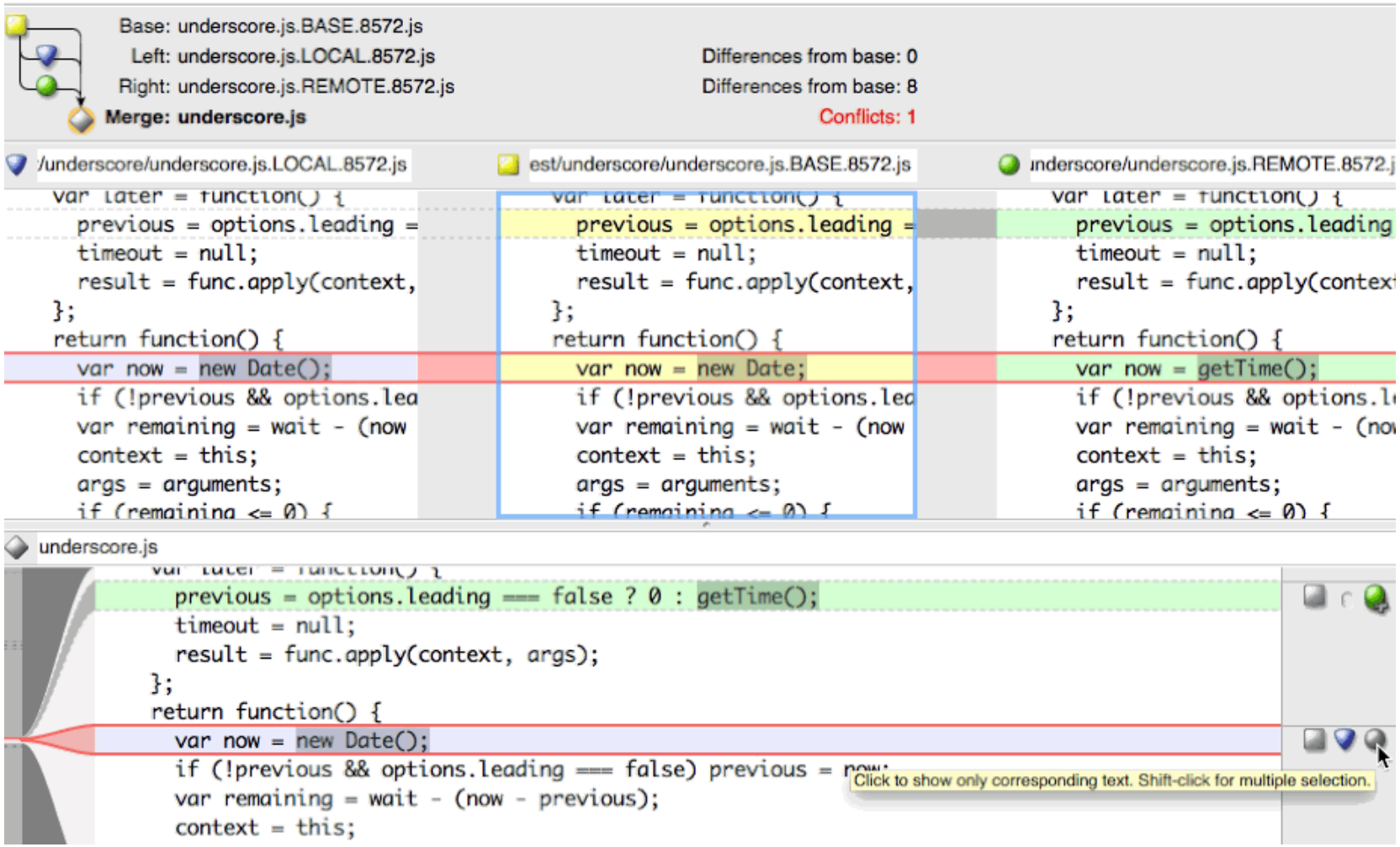
Nicht-linearer Ansatz: beide holen sich eine Version



Merge Conflict beim zweiten Commit

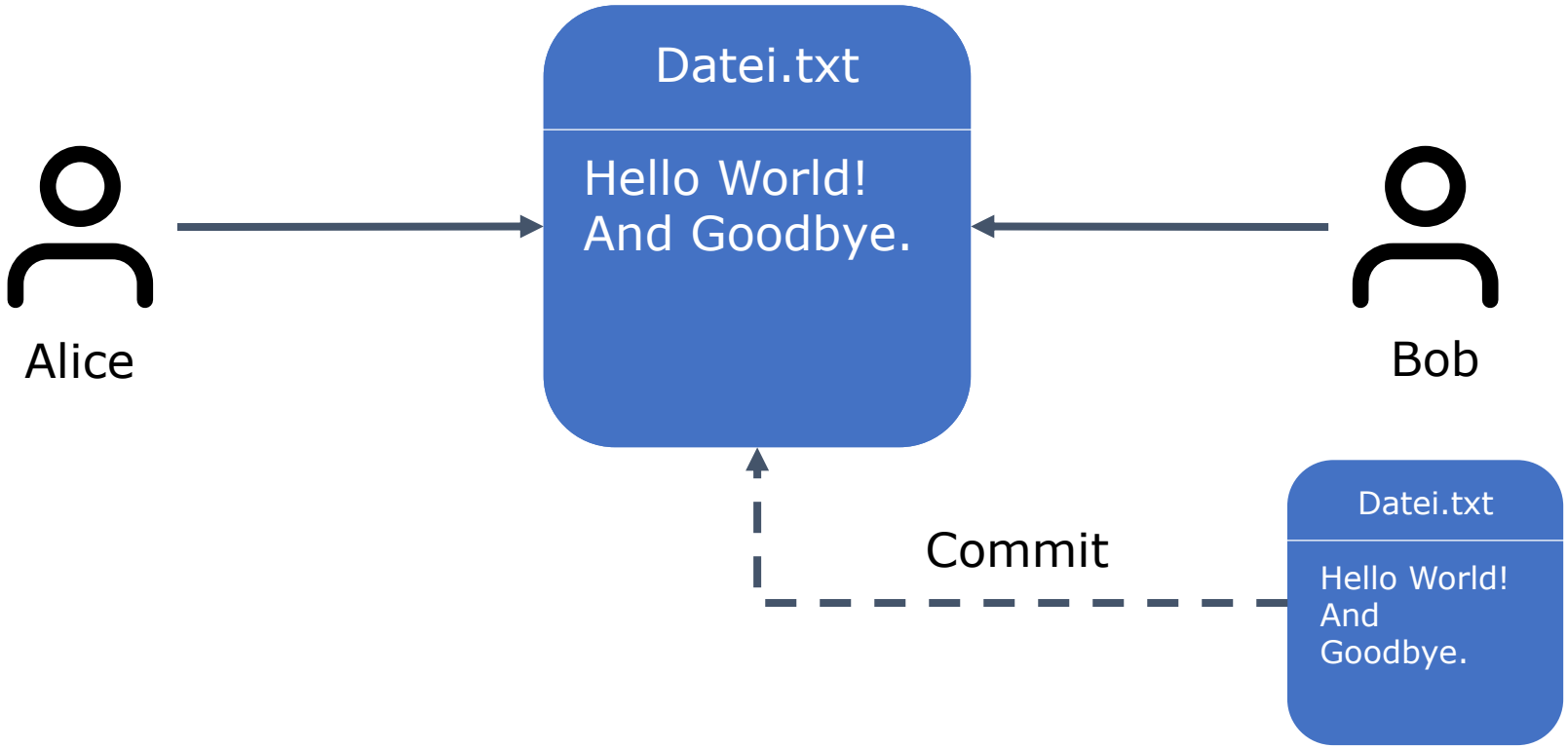


Einsatz eines Merge Tools wie p4merge zum Auflösen

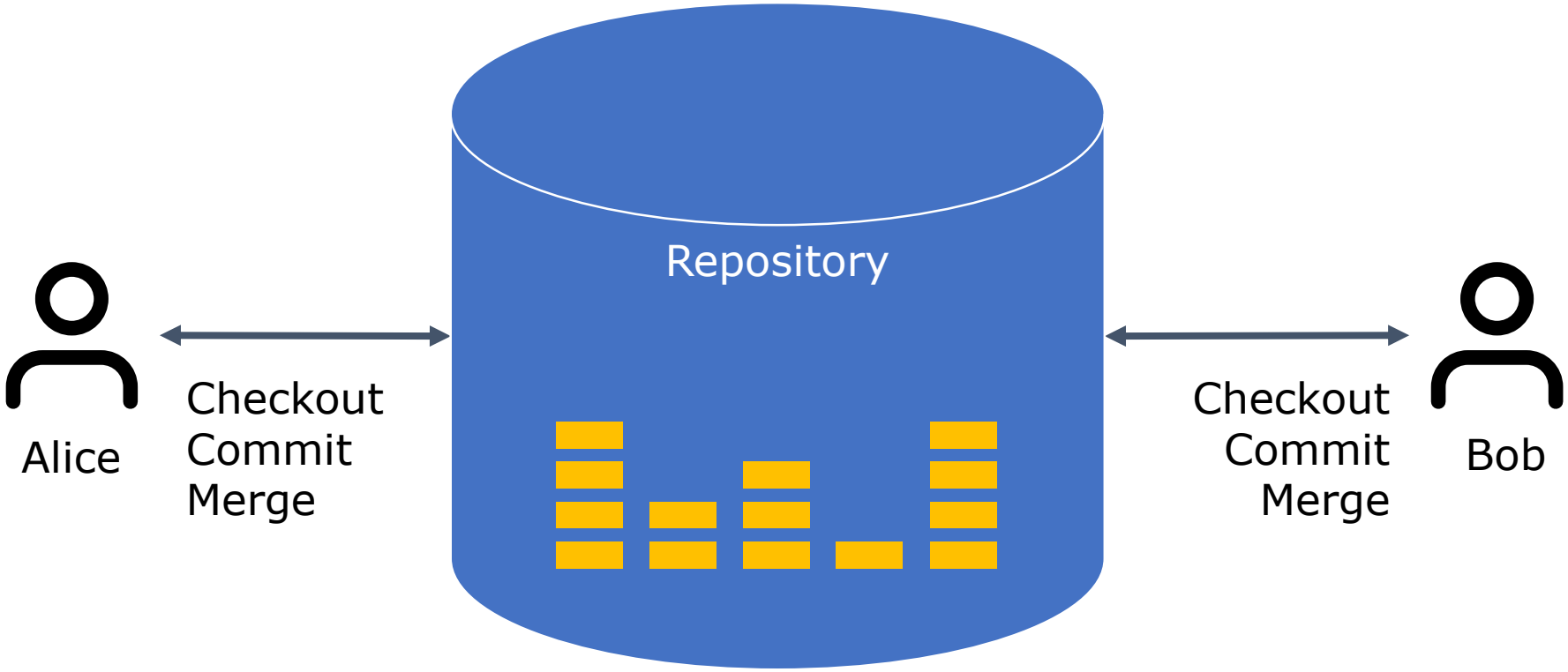


Quelle: http://naleid.com/images/four_pane_merge_p4merge.gif

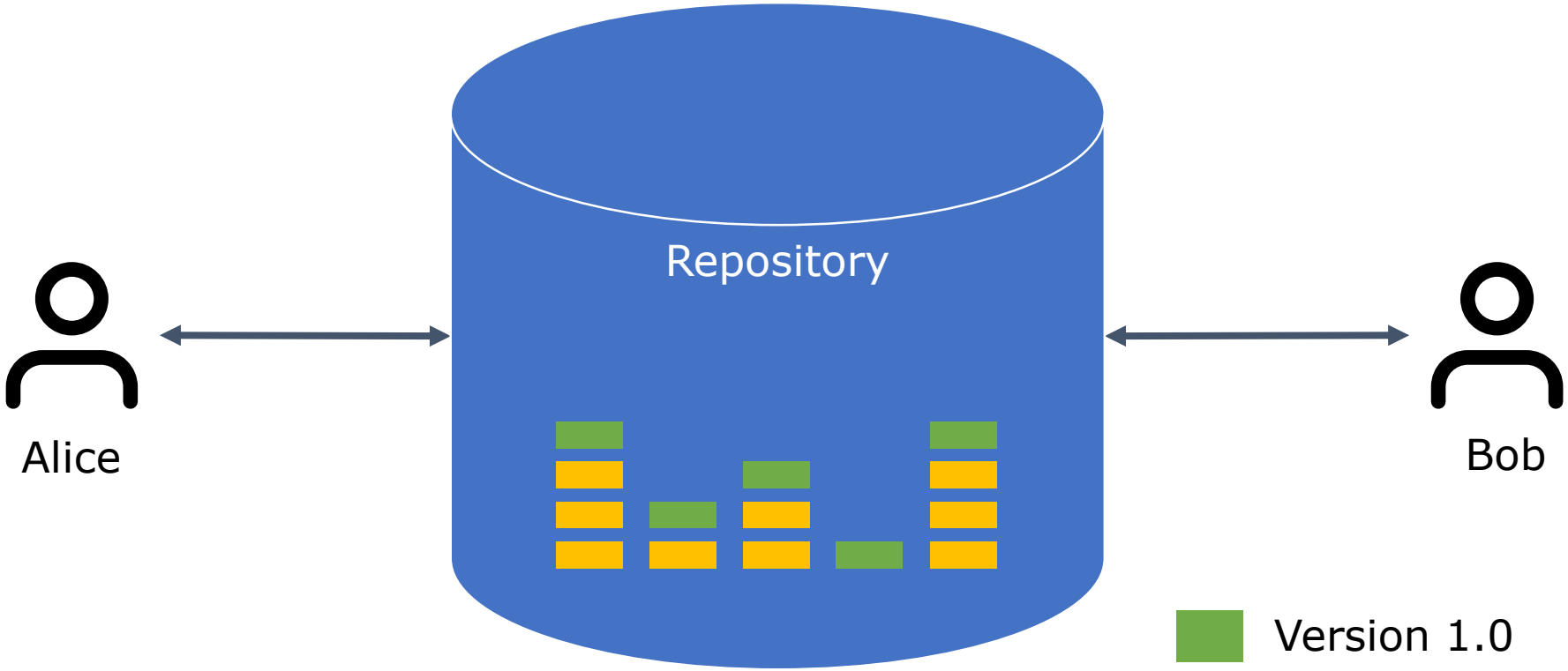
Anschließend Commit des aufgelösten Konflikts



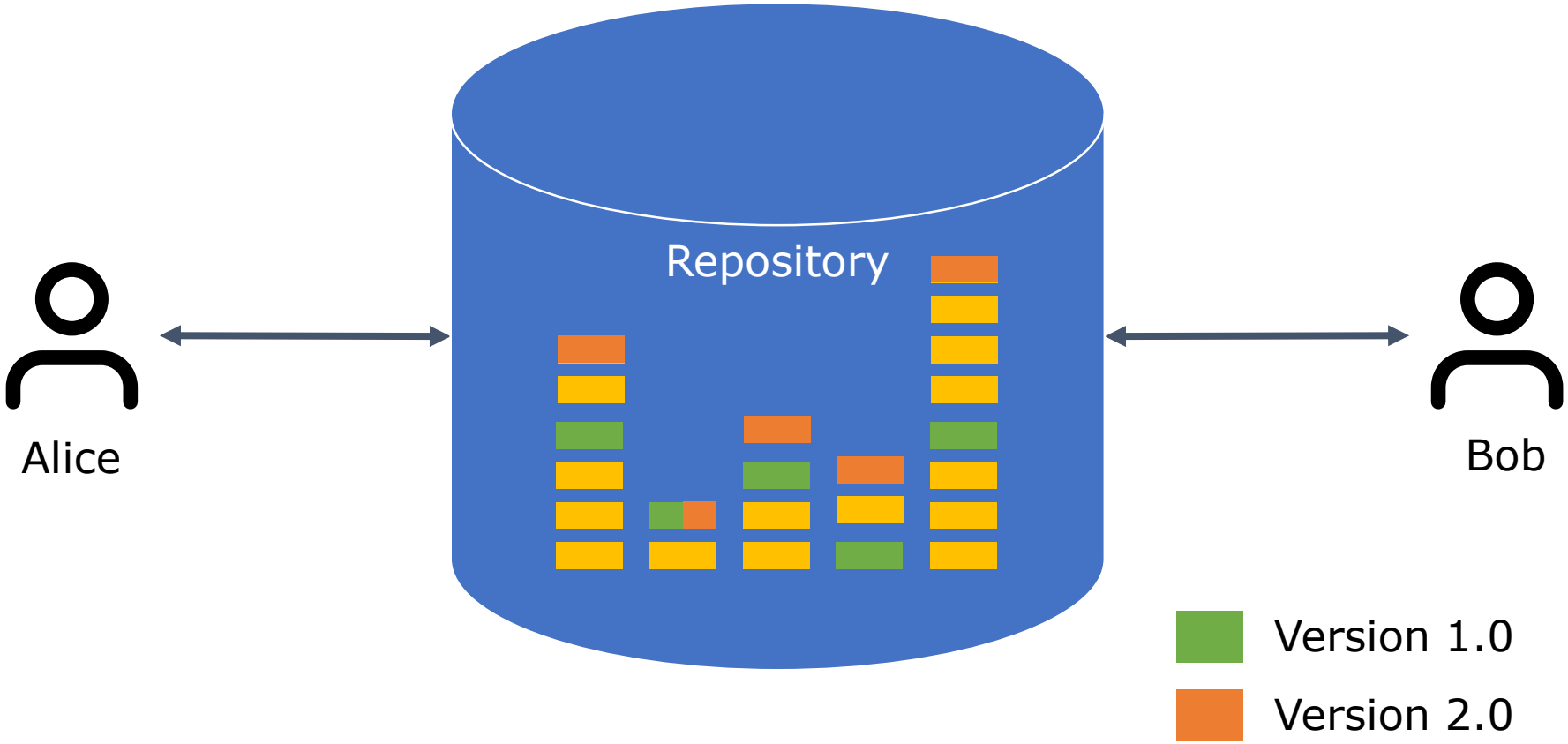
Zentralisierter Workflow



Versionierung über tags / labels



Versionierung über tags / labels



Agenda

- Motivation
- Zentralisierter Ansatz
- **Dezentralisierter Ansatz mit Git**
- Workflow Modelle
- Zusammenfassung

Dezentraler Ansatz

Prinzip:

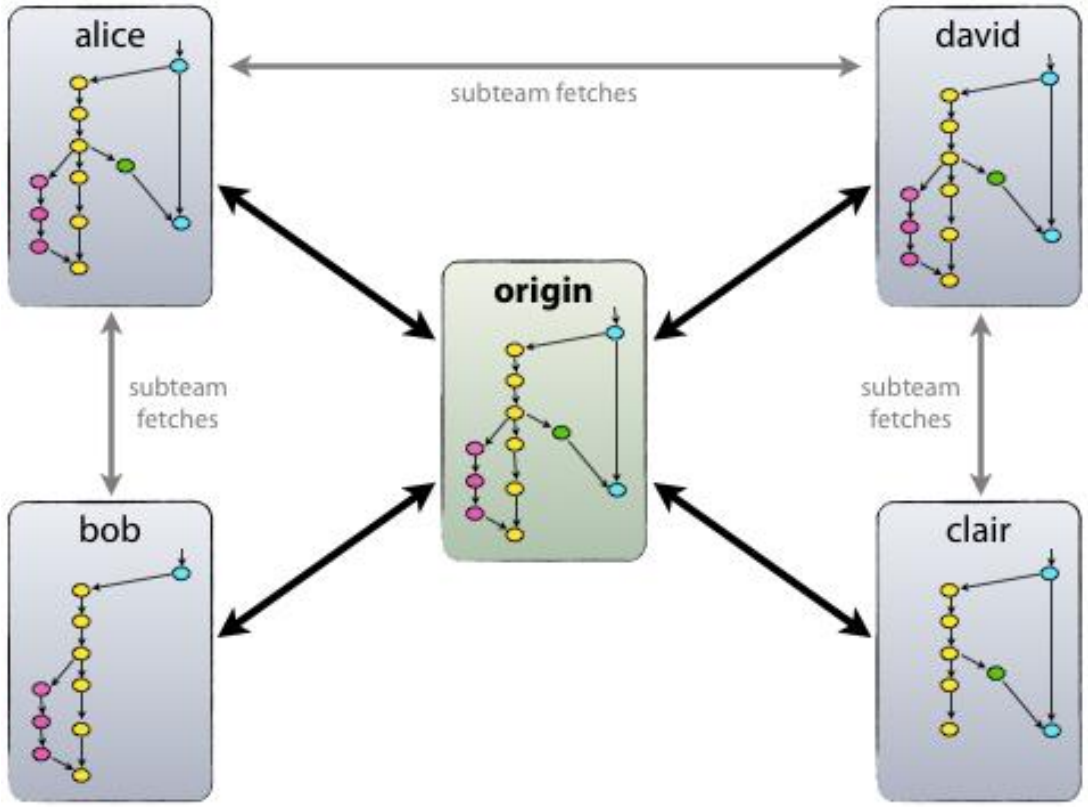
- Jeder Nutzer hat komplette Kopie des Repositories (hier origin)
- Austausch von Änderungen über Patches

Vorteile:

- Entkopplung ermöglicht unabhängiges und schnelleres Arbeiten
- Backup durch verteilte Redundanz
- Ermöglicht mehrere Workflows

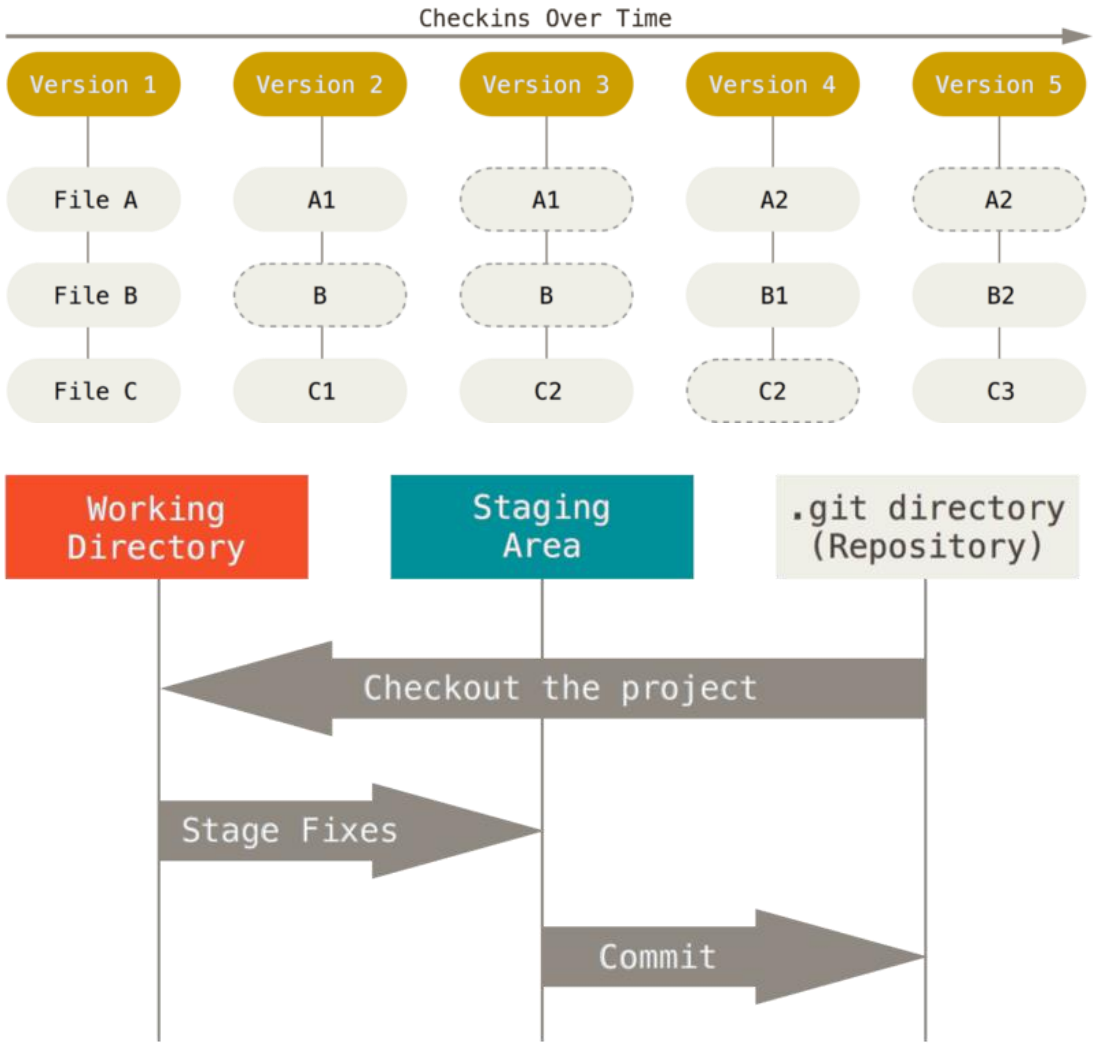
Nachteile

- Bedarf mehr Speicherplatz auf jedem Rechner
- Kein Locking-Mechanismus



Überblick Git

- 2005 von Linus Torvalds für Entwicklung des Linux Kernels entwickelt
- Ziele:
  - Schnell
  - Massiv verteilt
  - Einfaches Design
- Besonderes Feature: Branching sehr schnell!
- Versionierung basiert auf Snapshots des gesamten Filesystems (s. rechts oben)
- Lokale Operationen unterteilt in:
  - Checkout + Edit
  - Stage changes
  - Commit staged changes
- Separate Remote Operationen



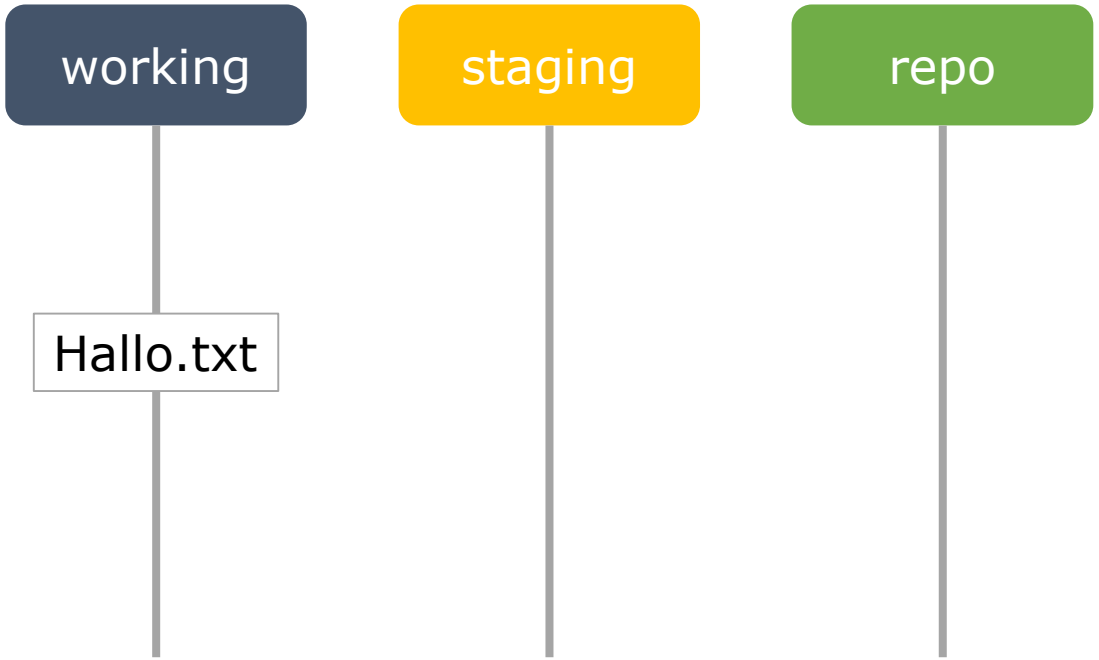
Quelle: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Git

Local Operations

Konfiguration und Initialisierung

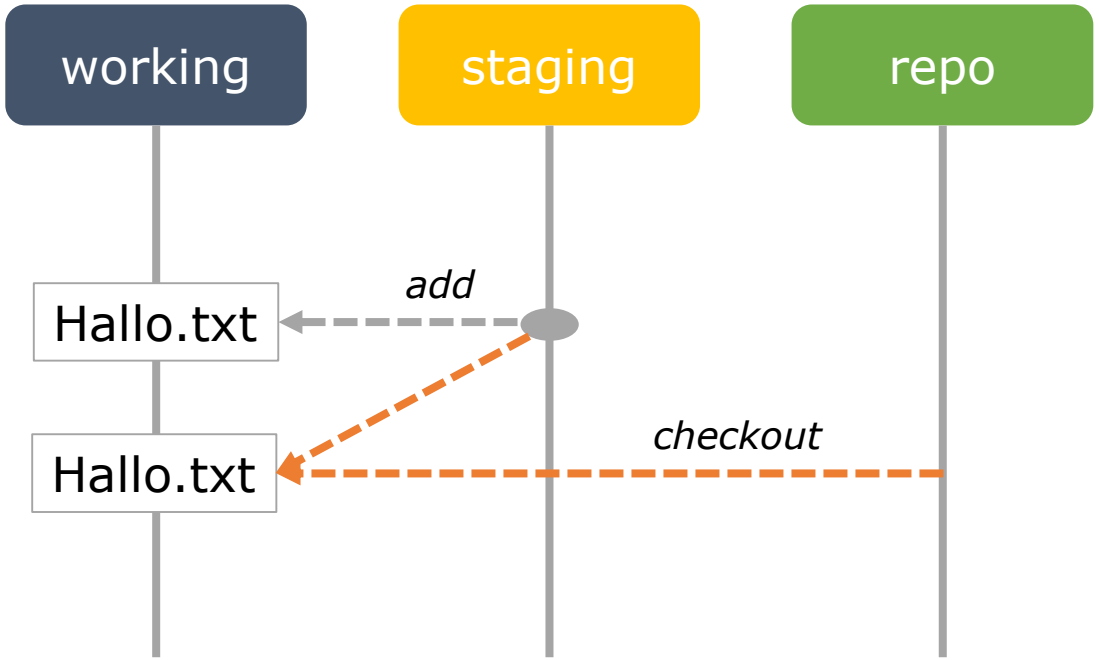
```
$ git config --global user.name "Your Name"
$ git config --global user.email "your_email@whatever.com"
$ git init
```



- Festlegen des Namens und der E-Mail Adresse
- Erstellung des lokalen Repositories im aktuellen Verzeichnis

Staging

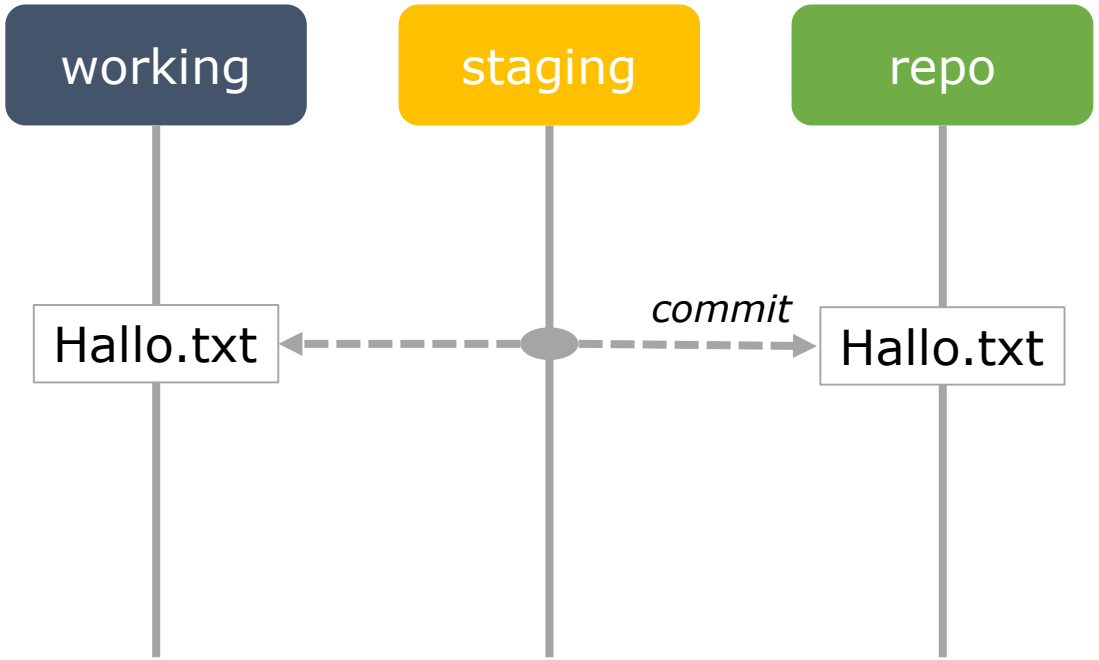
```
$ git add Hallo.txt
$ git checkout master
```



- Zum Index hinzufügen (stagen)
- Änderungen können über checkout rückgängig gemacht werden

Committing

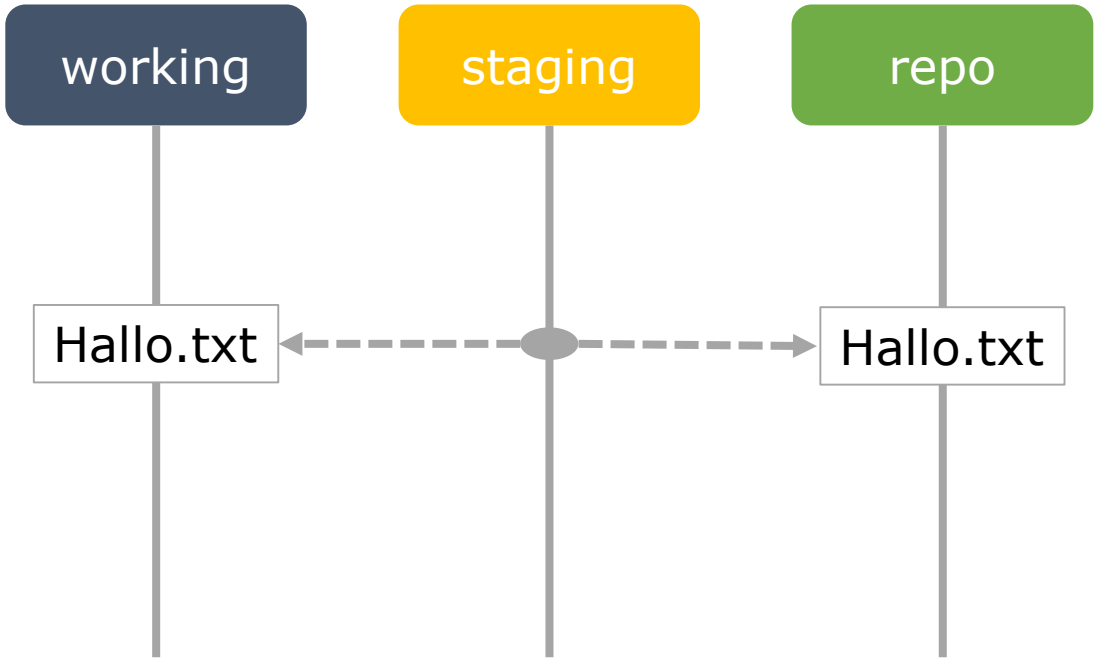
```
git commit –m "first Hallo"
```



- Schreibt die Änderungen in das lokale Repository
- Nur die Änderungen in der staging area werden committed!

Nützliche Befehle

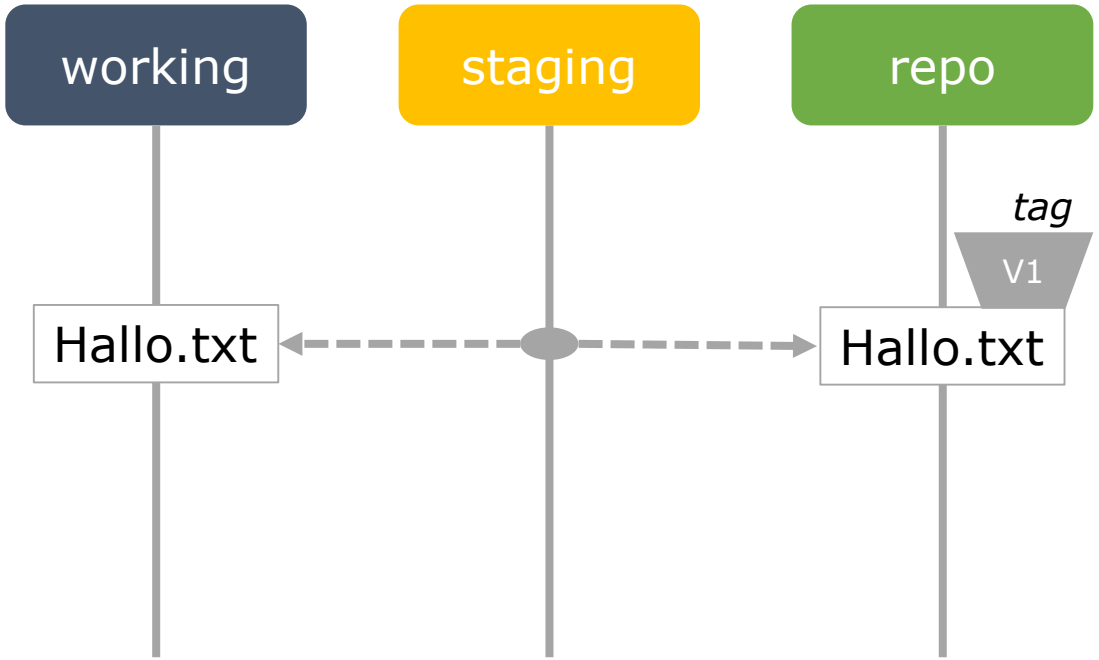
```
$ git status

# on branch master
Nothing to commit

$ git log
```



- Status des aktuellen Working Directories und der Staging Area ausgeben
- Historie der Commits ausgeben (lokal!)

Tagging

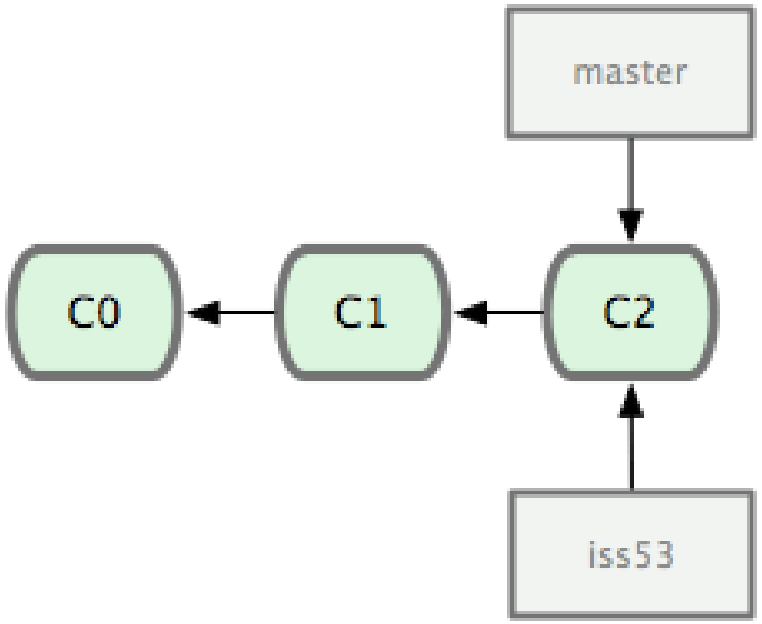
```
$ git tag v1
```



- Letzte Version im Repo mit Tag versehen

Branching

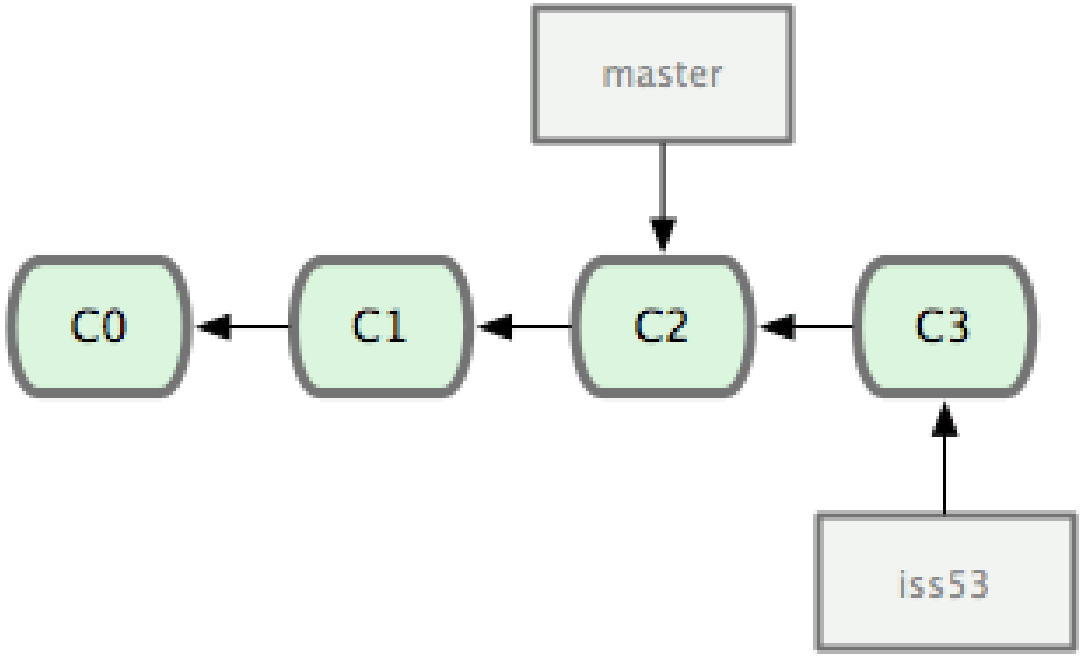
```
$ git branch iss53
$ git checkout iss53
```



- Branches können auch lokal angelegt werden
- Im wesentlichen Pointer auf einen Commit
- Wechsel der Branches schnell und einfach über checkout Befehl möglich
- Lokale Änderungen dürfen hierbei nicht existieren
- Best Practice: für jeden Task einen Branch anlegen und nutzen

Änderungen auf dem Branch

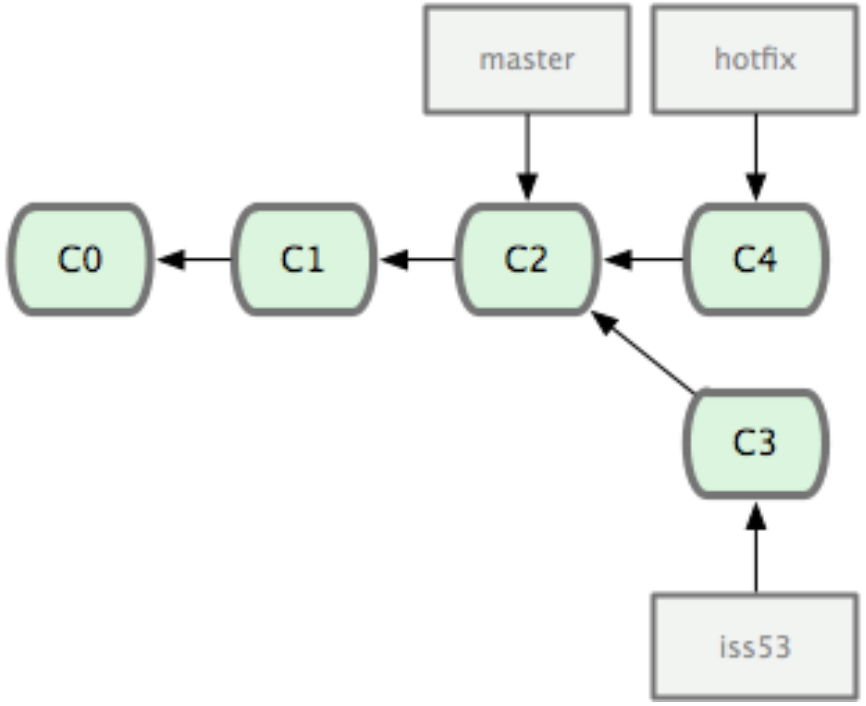
```
$ edit Hallo.txt
$ git commit –a –m „Änderung“
```



- Änderungen auf dem Branch mit anschließendem Commit verändern den “Zeiger” des aktuellen Branches
- Der Zeiger vom ursprünglichen Branch bleibt unberührt

Erstellung eines weiteren Branches

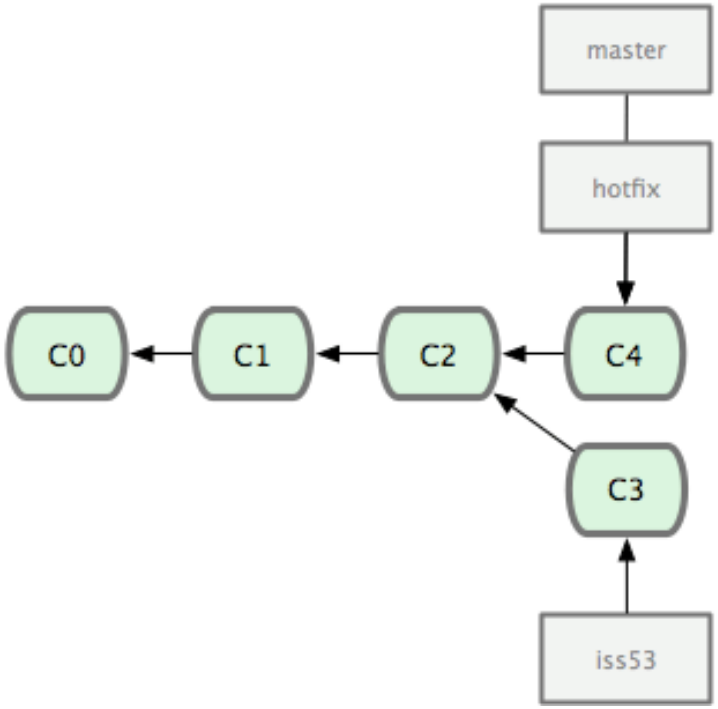
```
$ git checkout master
$ git checkout –b hotfix
$ edit Hallo.txt
$ git commit –a –m „Änderung“
```



- Änderungen auf anderen Branches führen zu weiteren Commits abseits des Ursprungs-Branches
- Mit checkout –b kann auf einen neuen Branch gleich gewechselt werden
- Mit commit –a werden alle Änderungen auch gestaged und dann committed

Mergen von Änderungen (fast-forward)

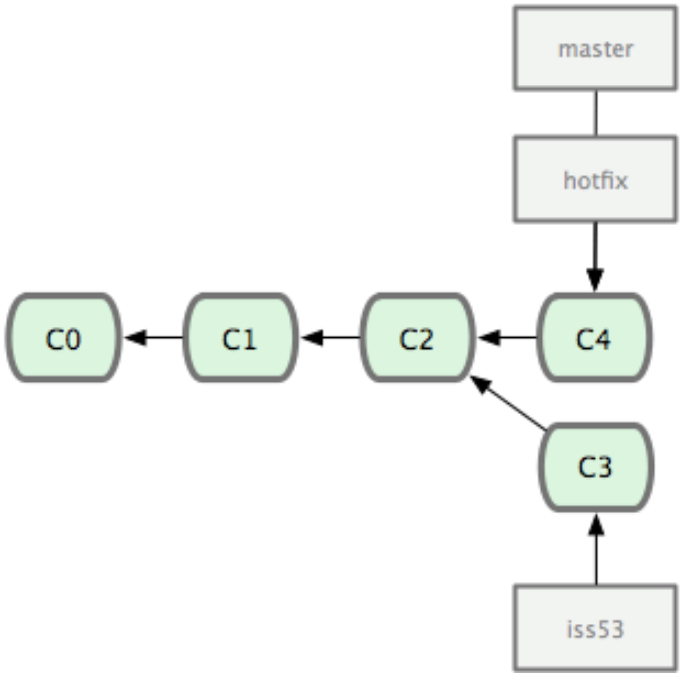
```
$ git checkout master
$ git merge hotfix
```



- Wechel auf den Ziel-branch
- Merge aller Änderungen über merge <branch>
- Führt alle Commits auf den aktuellen Stand des Ziel-Branches aus
- In diesem einfachen Fall: fast-forward (Versetzung des Zeigers)

Löschen eines Branches

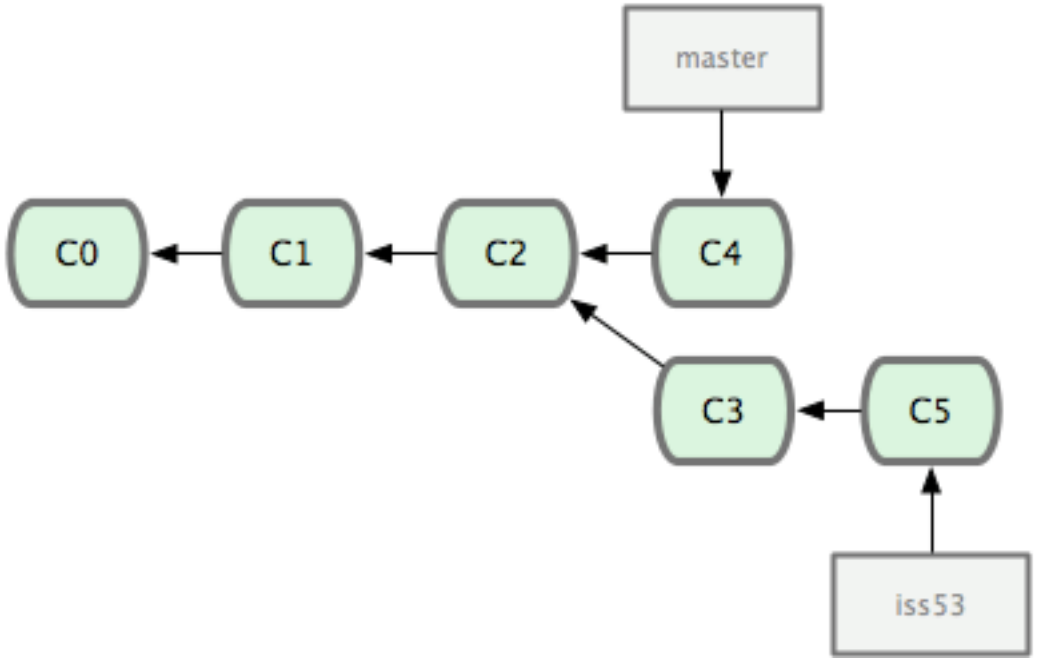
```
$ git branch -d hotfix
```



- Branches können einfach gelöscht werden
- Vorsicht: Historie geht verloren!
- In der Regel aber Best-Practice, um Übersicht über die Branches zu behalten
- Anschließend kann Ticket im Ticketsystem gelöscht werden (kann auch automatisiert geschehen)

Weiterführung des Beispiels

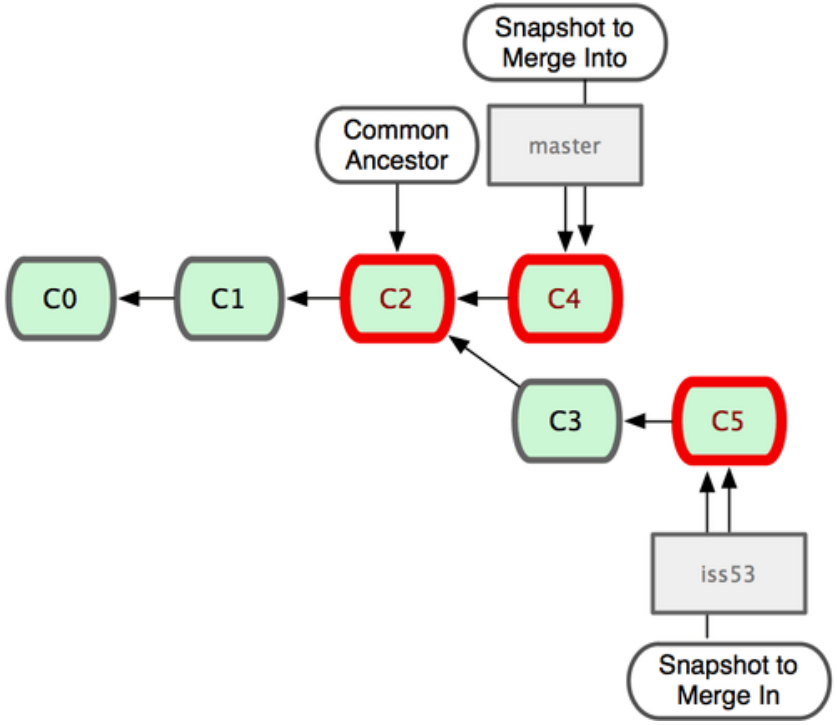
```
$ git checkout iss53
$ edit Hallo.txt
$ git commit –a –m “feature done“
```



- Wechsel auf iss53 Branch
- Editierien einer Datei
- Committen der Änderungen

Mergen von nicht-trivialen Änderungen

```
$ git checkout master
$ git merge iss53
```



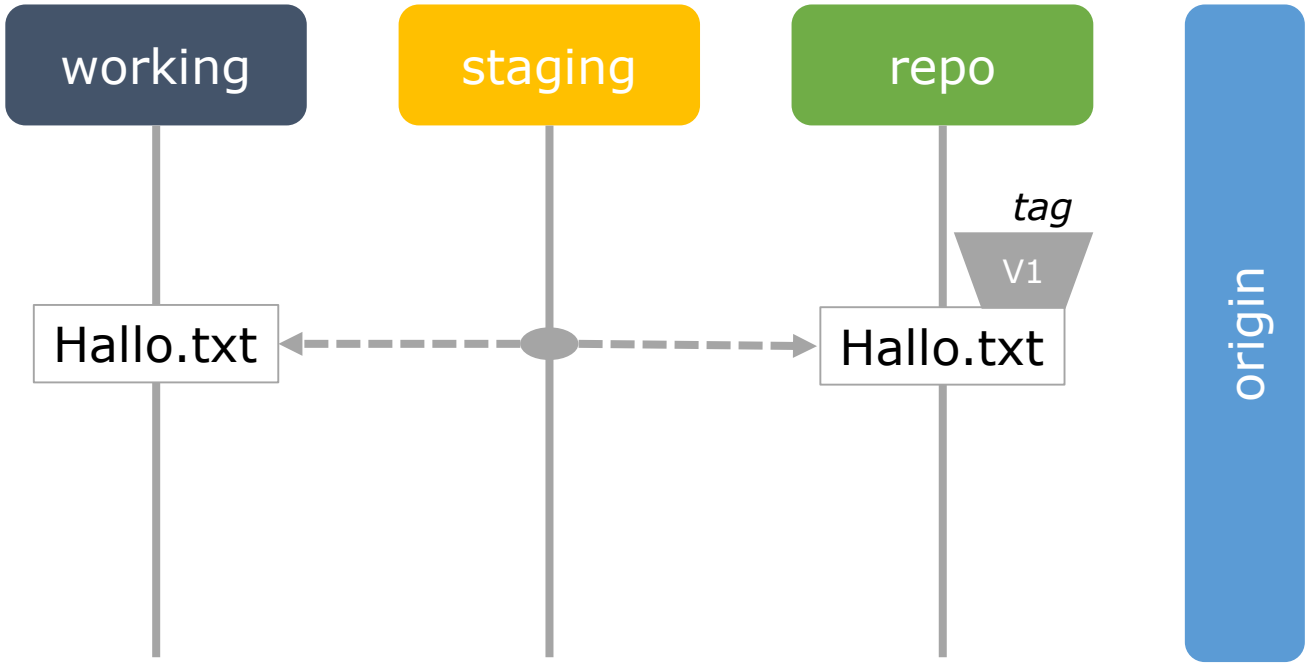
- Branches haben sich unterschiedlich weiterentwickelt
- Nun muss genau geprüft werden, ob es einen Merge-Konflikt (Änderung an selber Datei) gibt
- Wenn nötig, muss dieser Commit aufgelöst werden

Git

Remote Operations

Verknüfung mit Remote Repository

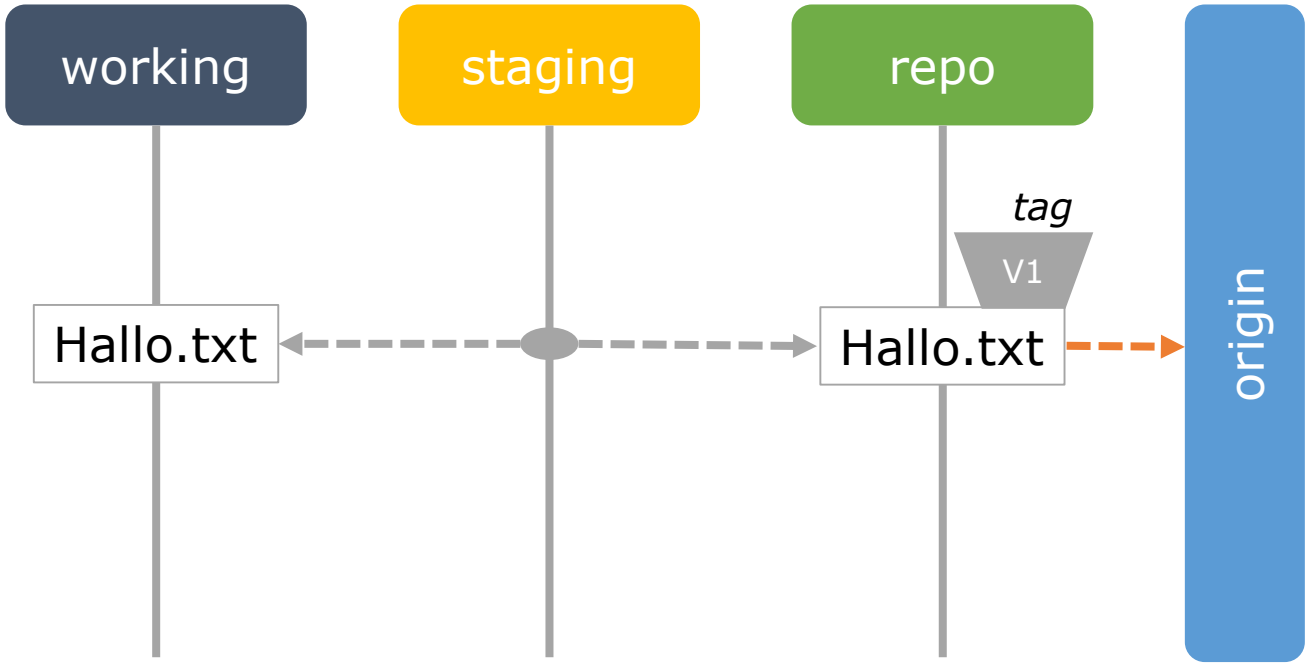
```
$ git remote add origin https://github.com
```



- Entfernes Repository hinzufügen
- Prinzipiell beliebig viele möglich denkbar
- Ermöglicht einfache Migration von einem zum anderen Repository

Remote Repository ändern

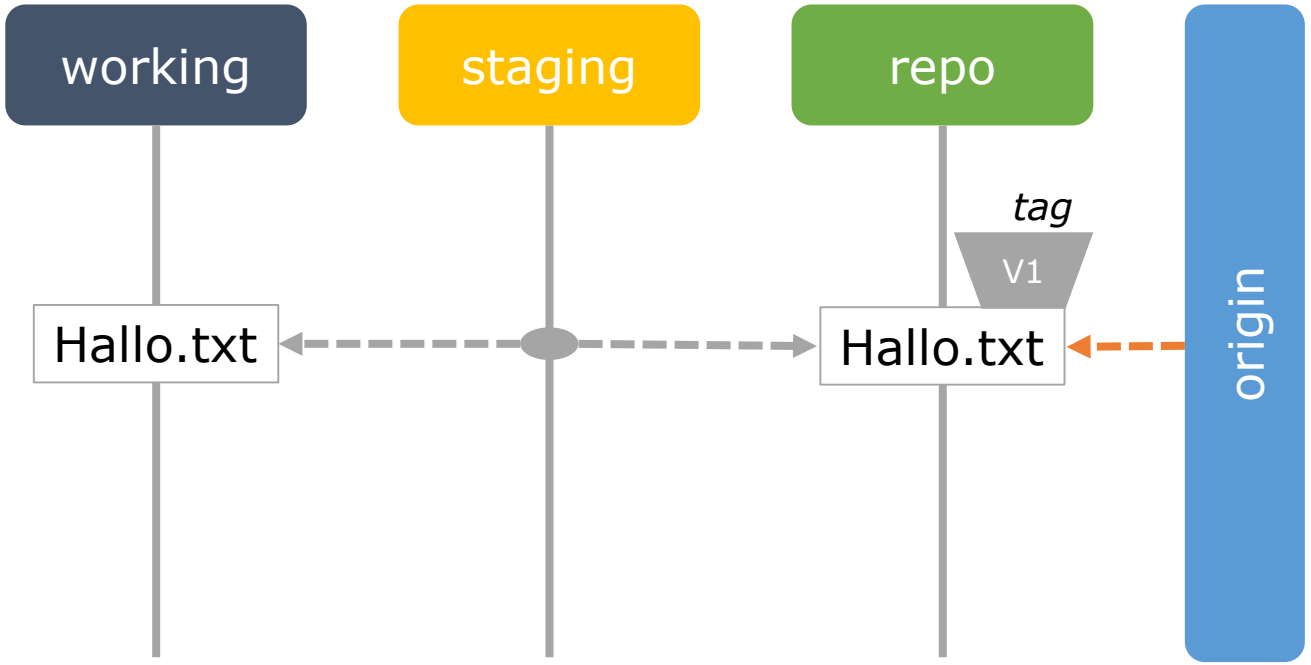
```
$ git push -u origin master
```



- Aktuellen Stand auf das remote repository schreiben

Änderungen vom Remote Repository abrufen

```
$ git pull origin master
```

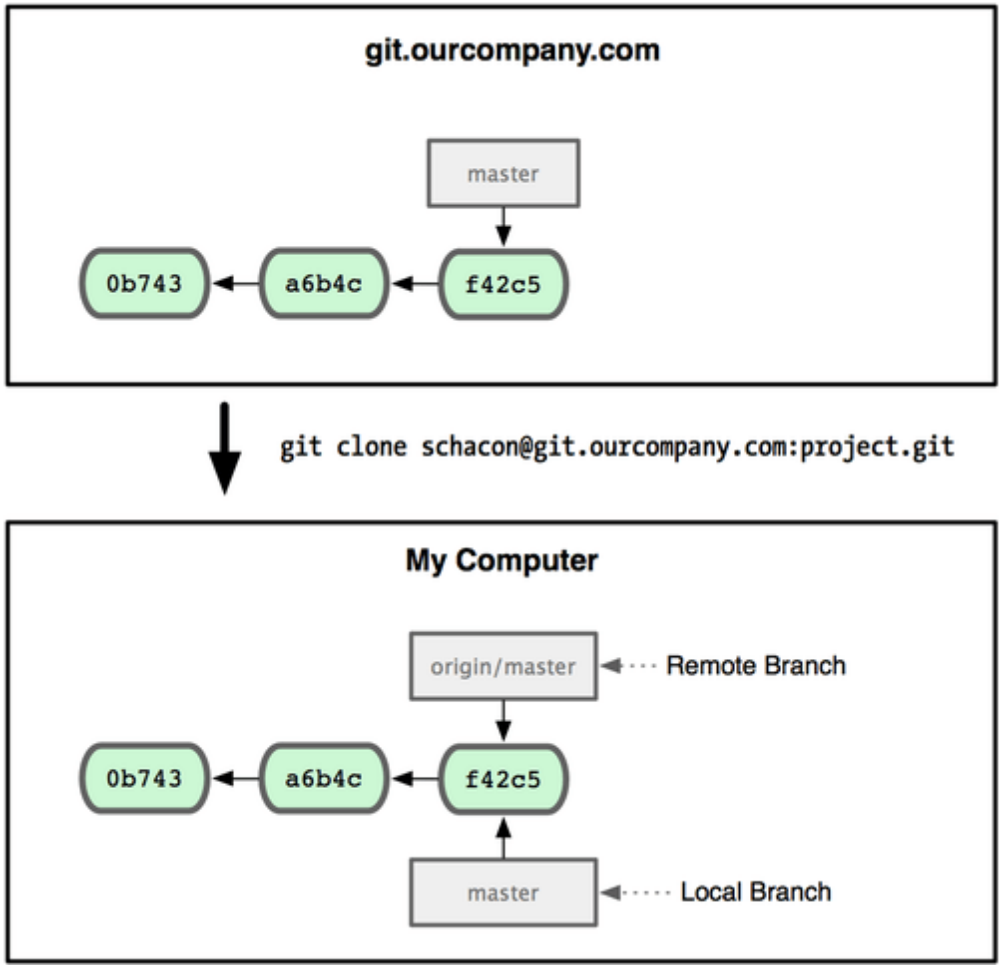


- Holt den aktuellen Stand aus dem Remote Repository
- Hier können Merge Konflikte auftreten!

Alternativ: Clonen eines Remote Repositories

```
$ git clone <url>
```

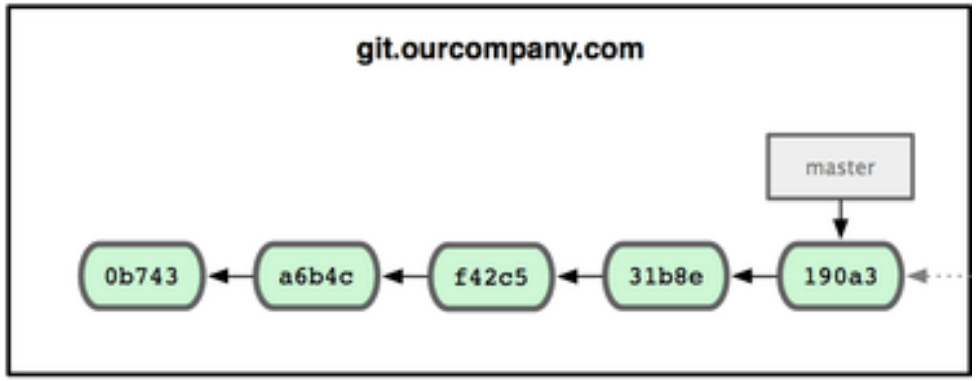
- Erstellt lokale Kopie des gesamten Repositories
- Fügt origin Repository automatisch hinzu
- Remote branches werden über *<repository>/<branch>* identifiziert

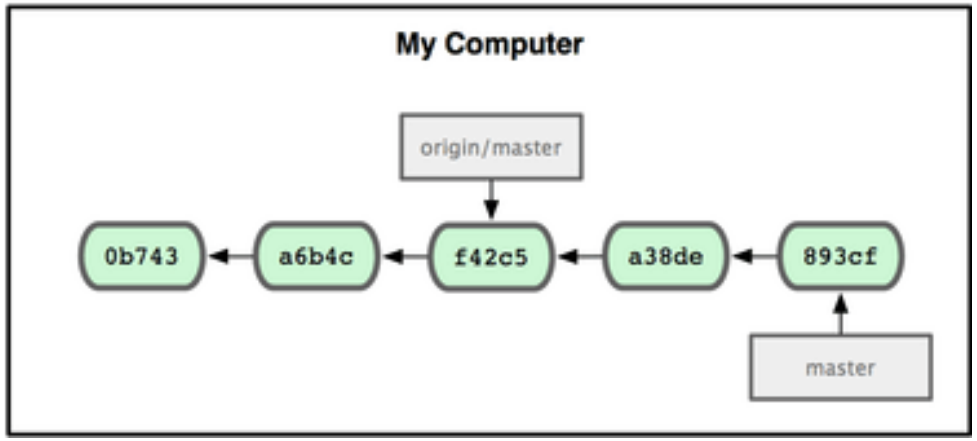


Lokaler Stand kann sich von Remote unterscheiden

```
$ edit Hallo.txt
```

- Lokal kann unabhängig gearbeitet und committed werden
- Das gleiche gilt für jeden weiteren Clone
- Remote können andere ebenso Änderungen pushen

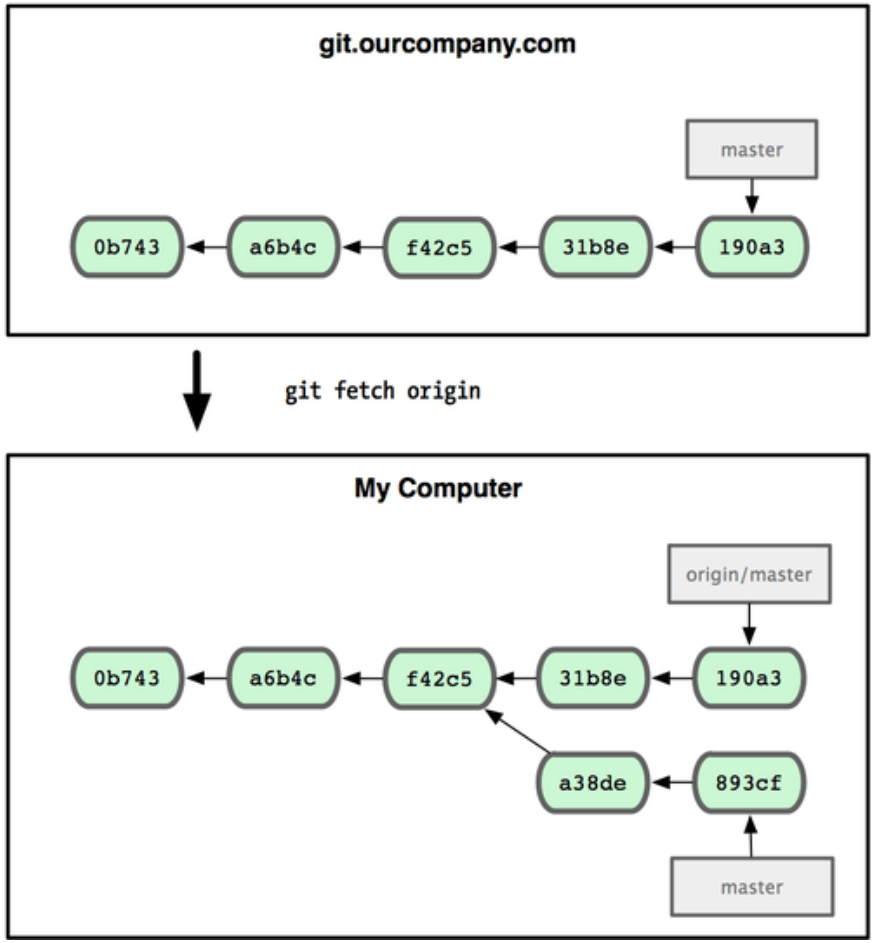




Fetch vs. Pull

```
$ git fetch origin
$ git pull
```

- Ruft Änderungen vom Remote Repository ab
- Lokale Branches bleiben hiervon zunächst unberührt, d.h. es gibt keine Merge Konflikte
- pull hingen ruft Änderungen ab und merged diese:
  - git fetch origin
  - git merge origin/master



Agenda

- Motivation
- Zentralisierter Ansatz
- Dezentralisierter Ansatz mit Git
- **Workflow Modelle**
- Zusammenfassung

Workflow Modelle für DVCS

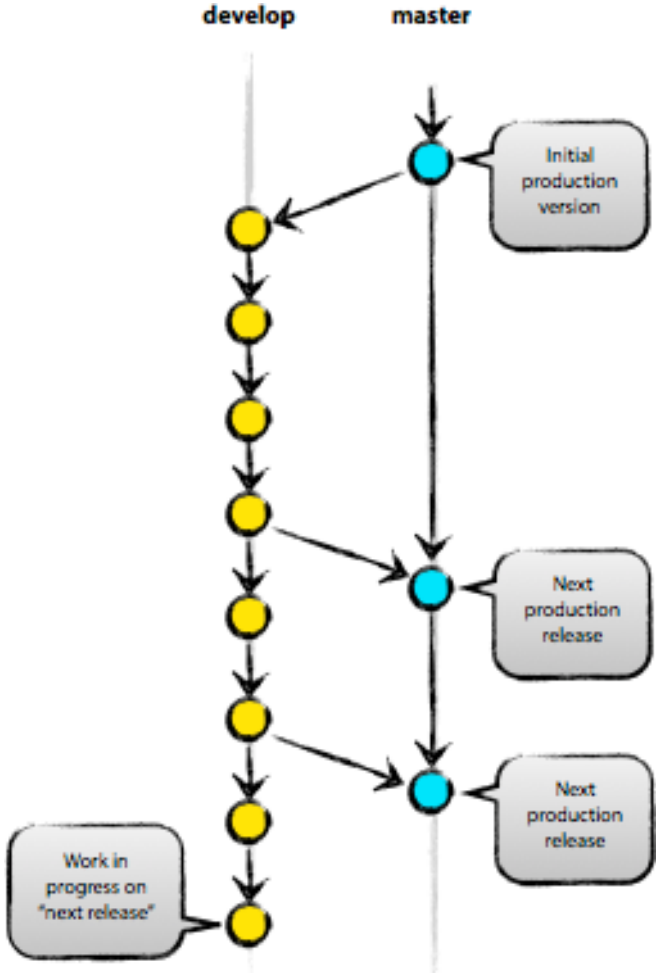
- Prinzipiell herrscht bei DVCS das Prinzip der Selbstorganisation
- In einem Team müssen jedoch Konventionen geschaffen werden, um die Entwicklung kontrollierbar und wartbar zu gestalten:
  - Releases?
  - Features?
  - Hotfixes?
  - Weiterentwicklungen?
- DVCS und auch Git bieten hier prinzipiell alle Freiheiten

Driessens Branch Modell

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

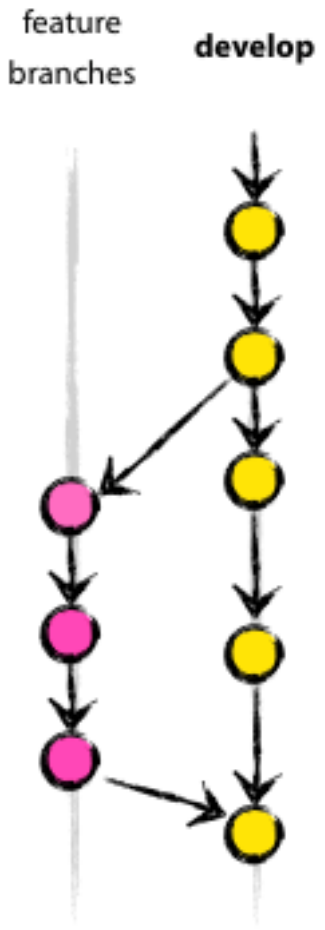
Master und Develop Branch

- Der Master Branch enthält eine stabile Version (master)
- Entwickelt wird auf einem Developer Branch (develop)
- Wenn die Entwicklung auf dem Developer Branch abgeschlossen
  ist, stabil läuft und ausgeliefert werden soll, wird auf den
  Master Branch gemerged
- Auslieferversionen auf dem Masterbranch werden mit Labels
  versehen (Tags).



Feature Branch

- Basierend auf einer lauffähigen Version wird ein Feature Branch erzeugt und genutzt
- Nach der Entwicklung werden die Änderungen der anderen Entwickler gemerged
- Wenn alles läuft, werden einmal wöchentlich alle Entwicklungen auf den Developer Branch gemerged, gelabelt, und ein Regressionstest gemacht.
- Diese Version nutzt die Qualitätssicherung in der Folgewoche
- Die Entwicklungsabteilung entwickelt das nächste Feature auf einem neuen Feature Branch
- Diese Vorgehensweise erlaubt das gezielte Versionieren einzelner Features.
- Durch wöchentliches Build wird früh integriert. Dadurch werden mögliche Fehler früh entdeckt.

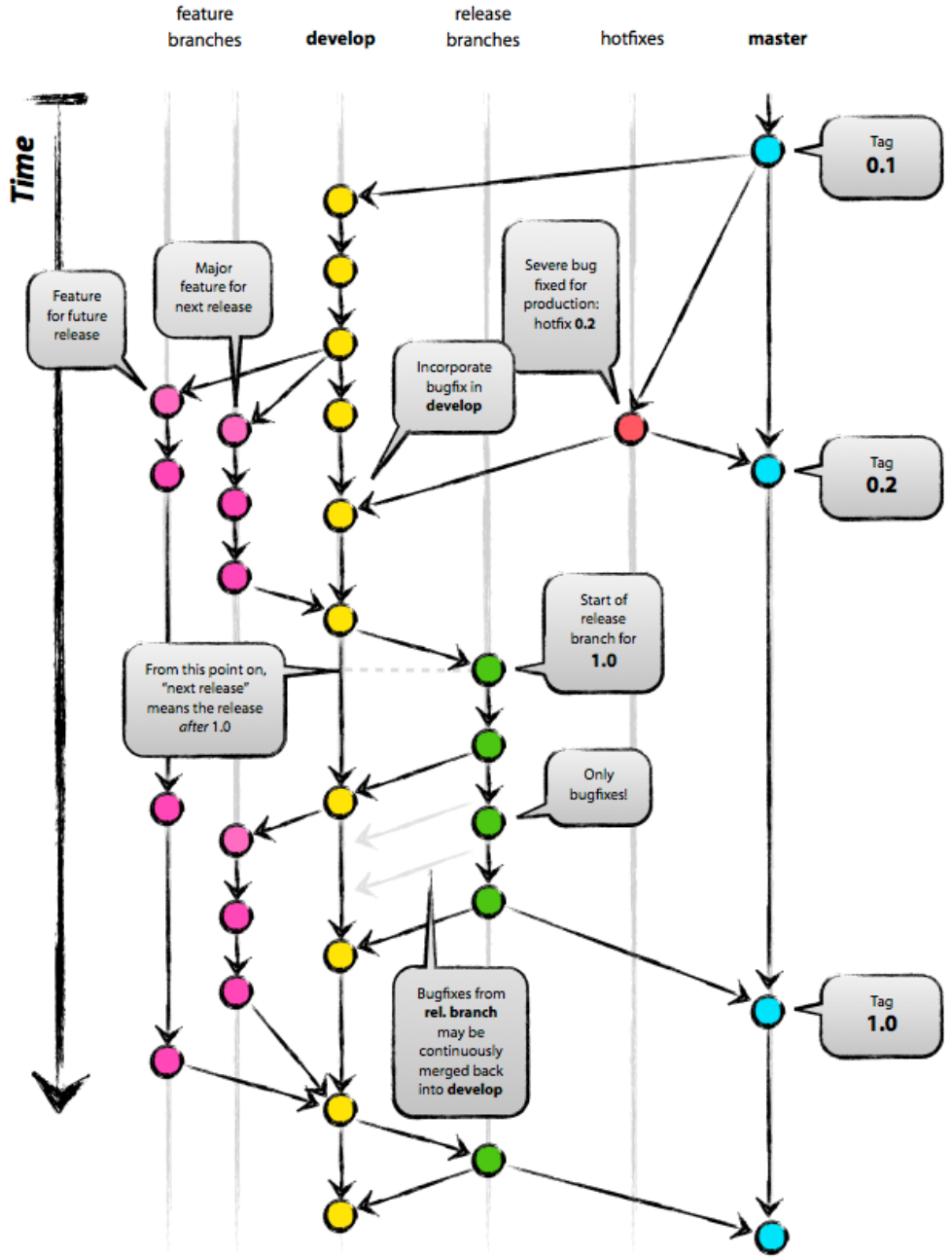


Hotfix Branch

- Dringende Fehler (Show-Stopper) müssen sofort repariert werden
- Dies geschieht auf einem Hot Fix Branch
- Die Reparatur wird auch in Folgeversionen und den aktuellen Developer Branch gemerged

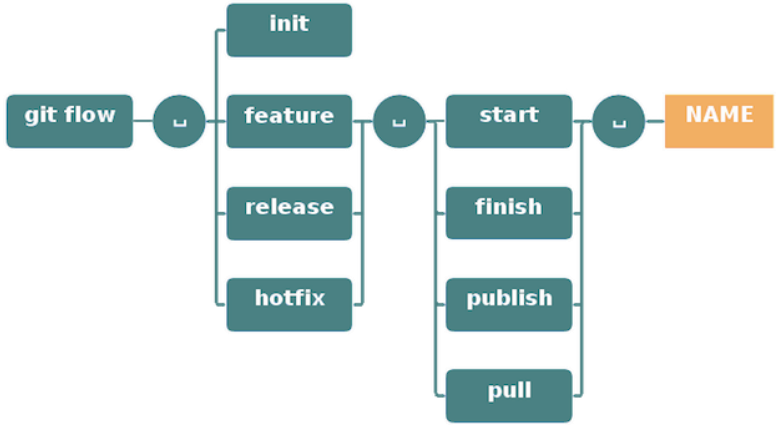
Release Branch

- Ein Release wird sehr lange getestet
- Oft geht die Entwicklung für zukünftige Releases parallel weiter (develop)
- Fehler im Release werden auf den Developer Branch gemerged, damit diese Fehler nicht in zukünftigen Releases wieder drin sind.



Git flow

- git-flow bietet high-level git-Operationen, die Driessens Branching Modell unterstützen
- siehe auch folgendes Cheatsheet
- Viele grafische Tools unterstützen git flow ebenfalls (z.B. GitKraken)
- **git-flow erweitert git nicht, sondern ist „nur" ein Modell mit Konventionen für Branches!**

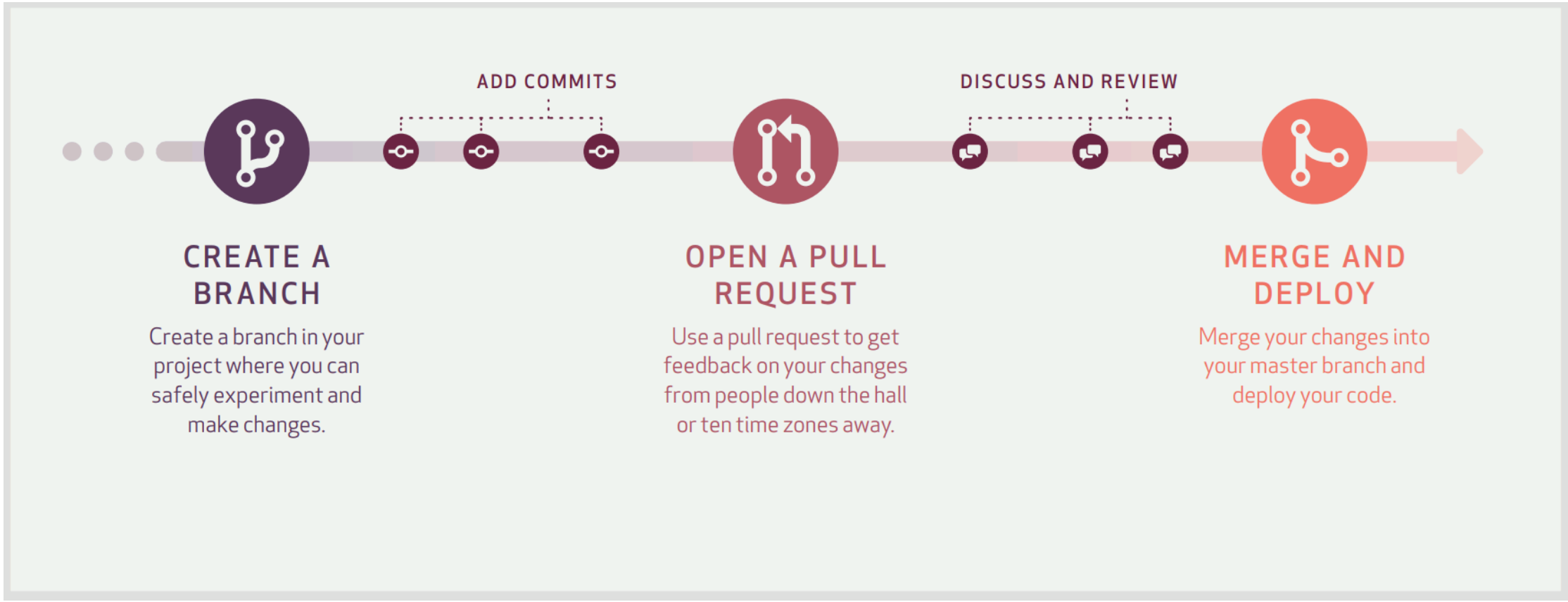



```
$ git flow init
$ git flow feature start MYFEATURE
$ git flow feature finish MYFEATURE
$ git flow release start RELEASE
  [BASE]
$ git flow release finish RELEASE
$ git flow hotfix start VERSION
  [BASENAME]
$ git flow hotfix finish VERSION
```

GitHub Flow a.k.a. Integration Manager Workflow

Quelle: https://guides.github.com/pdfs/githubflow-online.pdf

Leichtgewichtiger, branch-basierter Ansatz

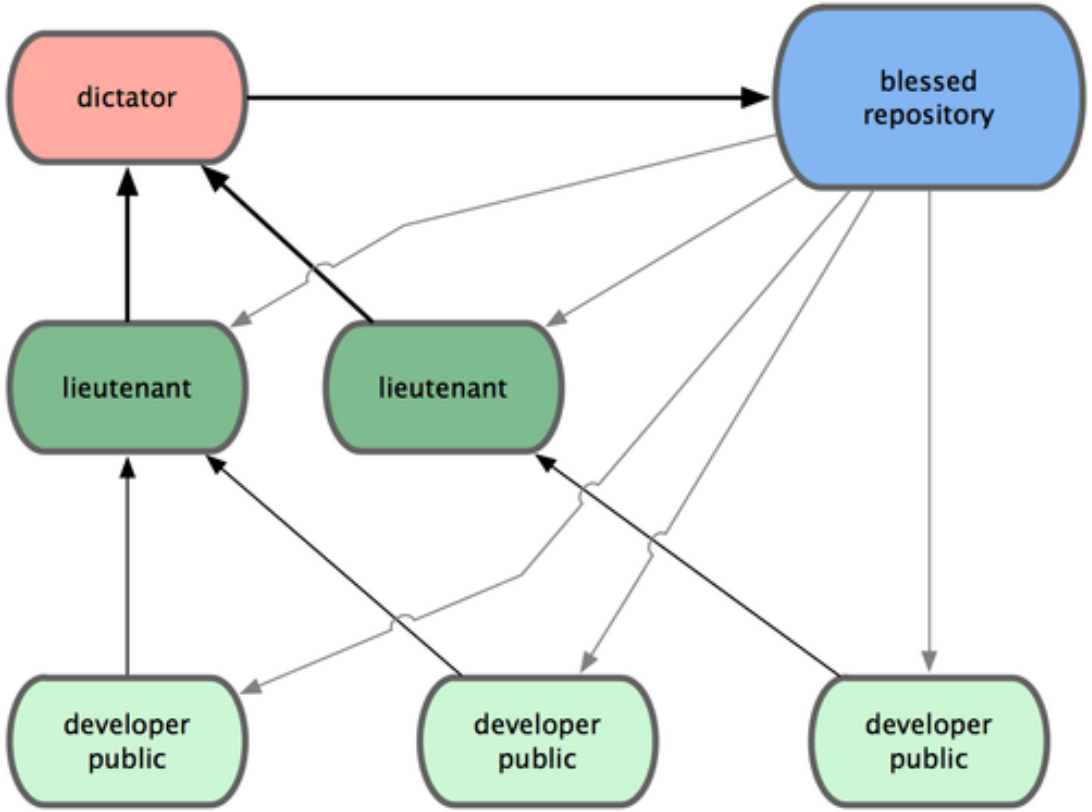


Diktator und Leutnant Workflow

Quelle: https://guides.github.com/pdfs/githubflow-online.pdf

Stark hierarchisierter Ansatz

- Normale Entwickler arbeiten in ihren Arbeitsbranches und synchronisieren ihre Änderungen auf der Basis des Master Branches. Der Master Branch ist derjenige des Diktators.
- Die Leutnants mergen die Arbeitsbranches der Entwickler in ihre Master Branches.
- Der Diktator merged die Master Branches der Leutnants mit seinem eigenen Master Branch zusammen.
- Der Diktator pusht seinen Master Branch ins Referenz-Repository, so dass alle ihre Arbeit wiederum damit synchronisieren können.



Agenda

- Motivation
- Zentralisierter Ansatz
- Dezentralisierter Ansatz mit Git
- Workflow Modelle
- **Zusammenfassung**

Zusammenfassung

- Versionsverwaltung ist unumgänglich in Projekten
  - Vermeidung von Chaos in der Zusammenarbeit
  - Verfolgung von Änderungen
  - Markieren von „wichtigen“ Versionen
  - usw.
- Unterscheidung
  - Zentralisierte Ansätze wie SVN, CVS, ...
  - Verteilte/dezentrale Ansätze wie GIT
- Bedeutung von Workflow Modellen für den verteilten Ansatz

Aufgabe

- Besprechen Sie in Ihrem Team, wie Sie die Versionswaltung gestalten wollen (10 min in jedem Team)
- Im Anschluß: kurze Vorstellung 5 min von jedem Team mit Begründung

Literatur

- Git Book
- GitHub Tutorial
- git - the simple guide
- Brief Introduction to Git (Slideshare)