

Advanced Software-Engineering

DHBW Stuttgart
28.10.2025

Janko Dietzsch

„Aktuelles“ – KI und der IT-Arbeitsmarkt ...



Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence

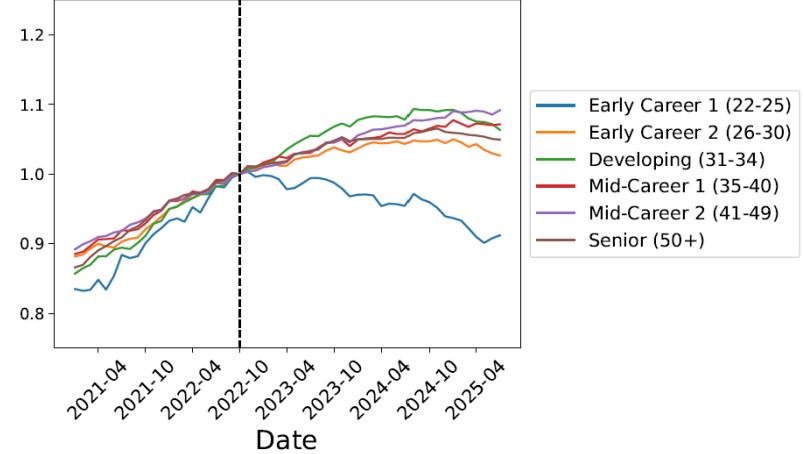
Erik Brynjolfsson*

Bharat Chandar†

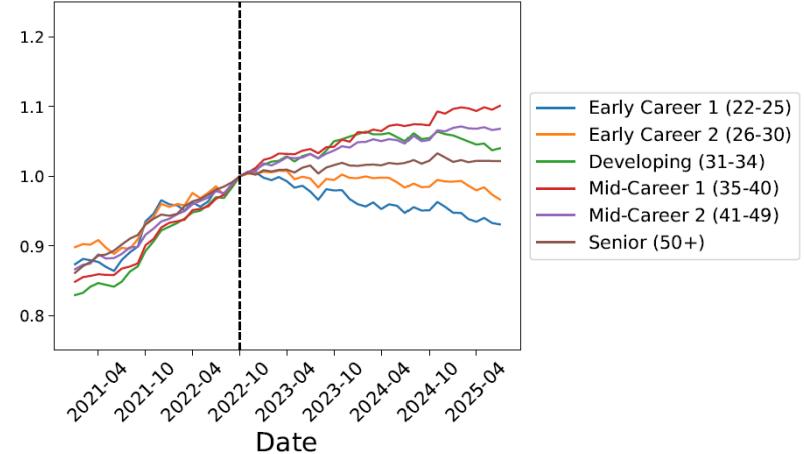
Ruyu Chen‡§¶

August 26, 2025

Headcount Over Time by Age Group
Computer Occupations (Normalized)



Headcount Over Time by Age Group
Service Clerks (Normalized)



„Aktuelles“ – KI und der IT-Arbeitsmarkt ...

Generative AI as Seniority-Biased Technological Change:

Evidence from U.S. Résumé and Job Posting Data*

Seyed M. Hosseini[†]

Guy Lichtinger[‡]

Preliminary
August 2025

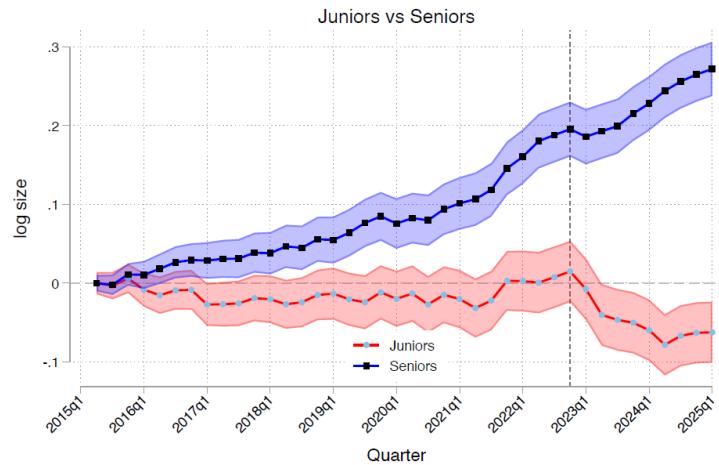


Figure 3: Employment Differences Between Adopters and Non-Adopters Over Time

Notes: The graph present the estimated coefficients β_j from Equation 1, ran separately to juniors and seniors. Standard errors are clustered in firm level.

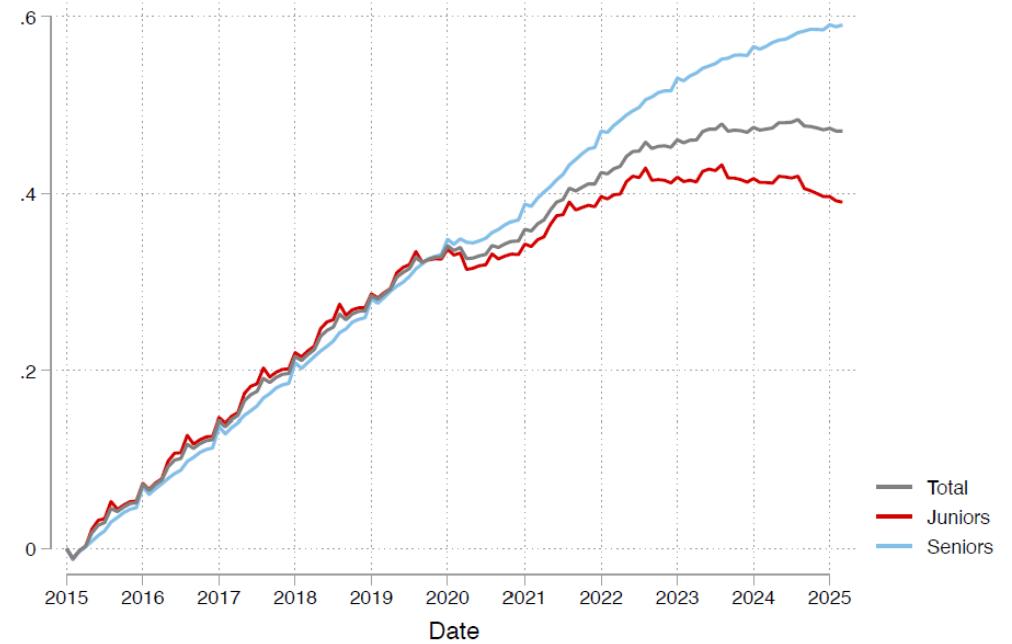


Figure 2: Time Series of Junior and Senior Employment in Sample Firms

Notes: This figure plots the average number of junior-level workers and senior-level workers in our sample of firms over time, normalized to 1 in January 2015. We define “junior” workers as those in Entry- or Junior-level positions, and “senior” workers as Associate level and above (see Section 3.1 for details).

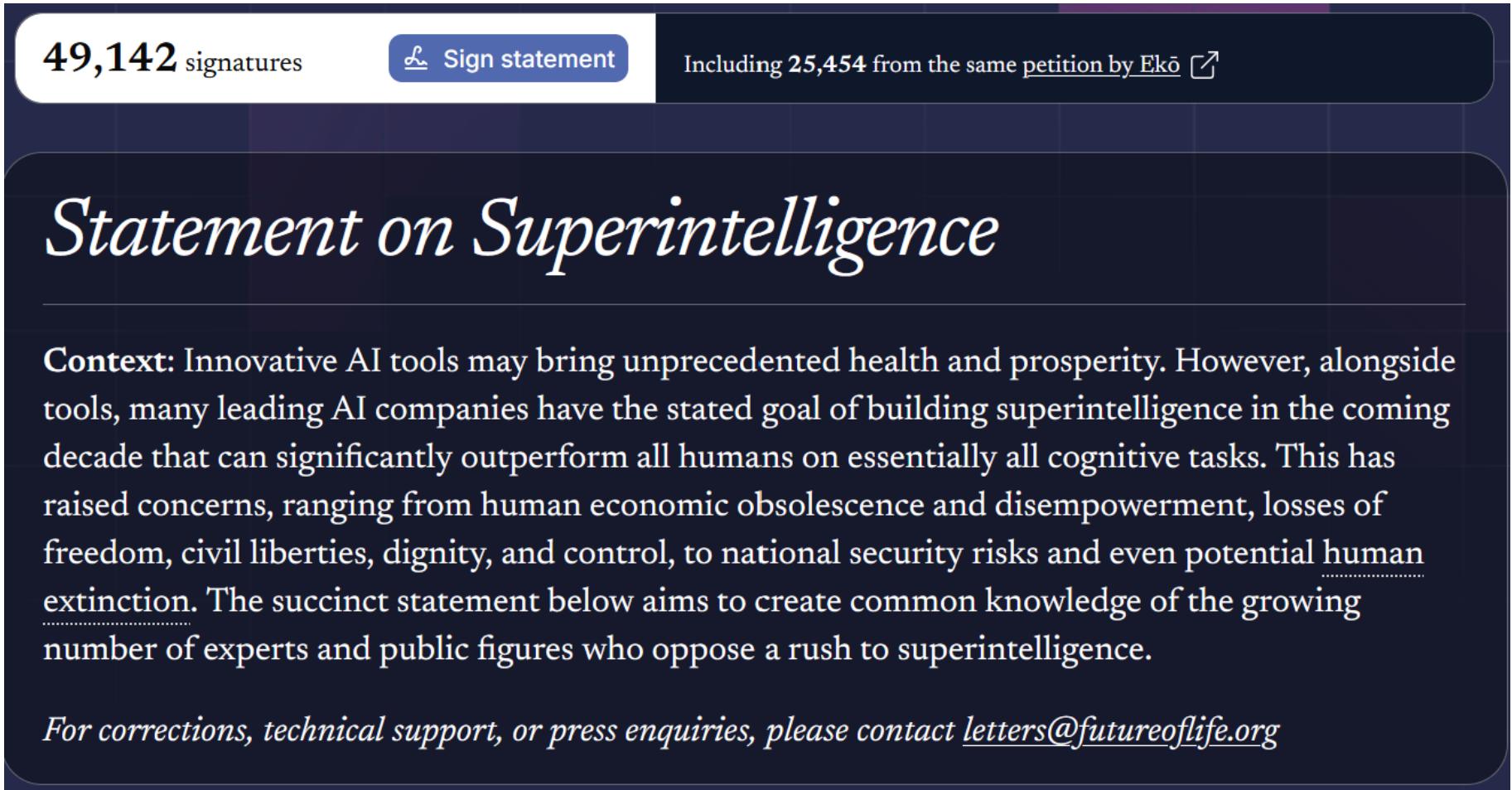
„Aktuelles“ – KI und der IT-Arbeitsmarkt ... und vielleicht nicht nur der ...



Godfather of AI: They Keep Silencing Me But I'm Trying to Warn Them!

The AI Safety Expert: These Are The Only 5 Jobs That Will Remain In 2030! - Dr. Roman Yampolskiy

„Aktuelles“ – Aufruf zur Regulierung von „Superintelligenz“



49,142 signatures [Sign statement](#) Including 25,454 from the same petition by Ekō ↗

Statement on Superintelligence

Context: Innovative AI tools may bring unprecedented health and prosperity. However, alongside tools, many leading AI companies have the stated goal of building superintelligence in the coming decade that can significantly outperform all humans on essentially all cognitive tasks. This has raised concerns, ranging from human economic obsolescence and disempowerment, losses of freedom, civil liberties, dignity, and control, to national security risks and even potential human extinction. The succinct statement below aims to create common knowledge of the growing number of experts and public figures who oppose a rush to superintelligence.

For corrections, technical support, or press enquiries, please contact letters@futureoflife.org

„Aktuelles“ – Aufruf zur Regulierung von „Superintelligenz“

Statement

We call for a prohibition on the development of superintelligence, not lifted before there is

1. broad scientific consensus that it will be done safely and controllably, and
2. strong public buy-in.

All Faith Leader Policymaker Arts & Media AI company Researcher Business

Search

Geoffrey Hinton

Emeritus Professor of Computer Science, University of Toronto, Nobel Laureate, Turing Laureate, world's 2nd most cited scientist

Yoshua Bengio

Professor of Computer Science, U. Montreal/Mila, Turing Laureate, world's most cited scientist

"Frontier AI systems could surpass most individuals across most cognitive tasks within just a few years. These advances could unlock solutions to major global challenges, but they also carry significant risks. To safely advance toward superintelligence, we must scientifically determine how to..."

Stuart Russell

Professor of Computer Science, Berkeley, Director of the Center for Human-Compatible Artificial Intelligence (CHAI); Co-author of the standard textbook 'Artificial Intelligence: a Modern Approach'

"This is not a ban or even a moratorium in the usual sense. It's simply a proposal to require adequate safety measures for a technology that, according to its developers, has a significant chance to cause human extinction. Is that too much to ask?"

Steve Wozniak

Co-founder of Apple

Sir Richard Branson

Founder, Virgin Group

Steve Bannon

Fmr Executive Chairman of Breitbart News; fmr chief strategist to President Donald Trump; Host of War Room podcast

Glenn Beck

Founder of Blaze media, radio host, TV personality, political commentator

Susan Rice

Fmr U.S. National Security Advisor under President Obama; U.S. Ambassador to the United Nations; Rhodes Scholar

Mike Mullen

U.S. Navy Admiral (ret), Chairman of the Joint Chiefs of Staff under Presidents George W. Bush and Barack Obama

Joe Crowley

Former Congressman (D) representing New York and House Democratic Caucus Chair

Key polling results on Superintelligence

5%

U.S. adults are in support of the status quo of fast, unregulated development

Polling Results 

64%

believe superhuman AI shouldn't be made until proven safe or controllable, or should never be made

73%

want robust regulation on advanced AI

Comments from signatories (Click to expand)

Yoshua Bengio

Professor of Computer Science, U. Mo...

Frontier AI systems could surpass most individuals across most cognitive tasks within just a few years. These advances could unlock solutions to major...

Sir Stephen Fry

Actor, director, writer

To get the most from what AI has to offer mankind, there is simply no need to reach for the unknowable and highly risky goal of superintelligence, which is b...

Mary Robinson

Fmr President of Ireland; Fmr UN Hi...

AI offers extraordinary promise to advance human rights, tackle inequality, and protect our planet, but the pursuit of superintelligence threatens to...

„Aktuelles“ – Bewegung bei Git



News from the source

Content

[Weekly Edition](#)

[Archives](#)

[Search](#)

[Kernel](#)

[Security](#)

[Events calendar](#)

[Unread comments](#)

[LWN FAQ](#)

[Write for us](#)

Edition

[Return to the Front page](#)

User: Password: [Log in](#) | [Subscribe](#) | [Register](#)

Git considers SHA-256, Rust, LLMs, and more

[LWN subscriber-only content]

Welcome to LWN.net

The following subscription-only content has been made available to you by an LWN subscriber. Thousands of subscribers depend on LWN for the best news from the Linux and free software communities. If you enjoy this article, please consider [subscribing to LWN](#). Thank you for visiting LWN.net!

By **Jonathan Corbet**

October 21, 2025

The Git source-code management system is a foundational tool upon which much of the free-software community is based. For many people, Git simply works, though perhaps in quirky ways, so the activity of its development community may not often appear on their radar. There is a lot happening in the Git world at the moment, though, as the project works toward a 3.0 release sometime in 2026. Topics of interest in the Git community include the SHA-256 transition, the introduction of code written in Rust, and how the project should view contributions created with the assistance of large language models.

„Aktuelles“ – Diskussionen in der Git-Entwicklergemeinde

- **Wechsel von SHA-1 zu SHA-256**
 - Seit 2.29 (2020) bereits Unterstützung für SHA-256
 - Problem ist aber die Interoperabilität alter SHA-1-mit SHA-256-Repo's
 - Angestrebt für Git 3.0 (~ 2026) ... aber noch sehr viel zu tun, um das zu erreichen
- **Rust-Verwendung**
 - Der Rust-Anteil an Entwicklung in Git steigt, z.B. in der Entwicklung für Interop. SHA-1/SHA-256, Reimplementierung der xdiff-Lib
 - Rust scheint ab Version 3.0 in der Entwicklung obligatorisch zu werden

„Aktuelles“ – Diskussionen in der Git-Entwicklergemeinde

- „**LLM-Code**“ und Git
 - Nach den Erfahrungen des **Google-Summer-of-Code 2025** mit großen Anteilen an KI-generierten Vorschlägen, sehr restriktive Fassung der Strategie
 - Diskussion ob zu streng
- Umstieg von **master** auf **main**
 - Für den Umstieg auf main als Default-Branch wird die Version 3.0 anvisiert
 - Sorge darum, dass das eine Vielzahl von Tutorien upgedatet werden müßte und vor allem Anfänger so mehr verwirrt werden könnten

„Aktuelles“ – Hilfe bekommen ...

↖(ᵔ⌇ᵔ)↖
IT WORKS
on my machine

Programmer Humor

@PROGRAMMERHUMOR

...

Newton's fourth law [reddit.com/r/programmerhu...](https://reddit.com/r/programmerhumor)

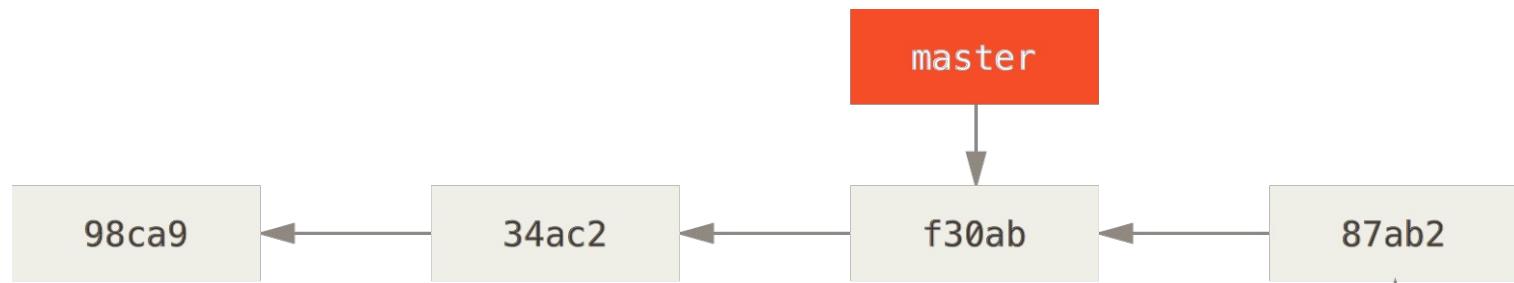


annie

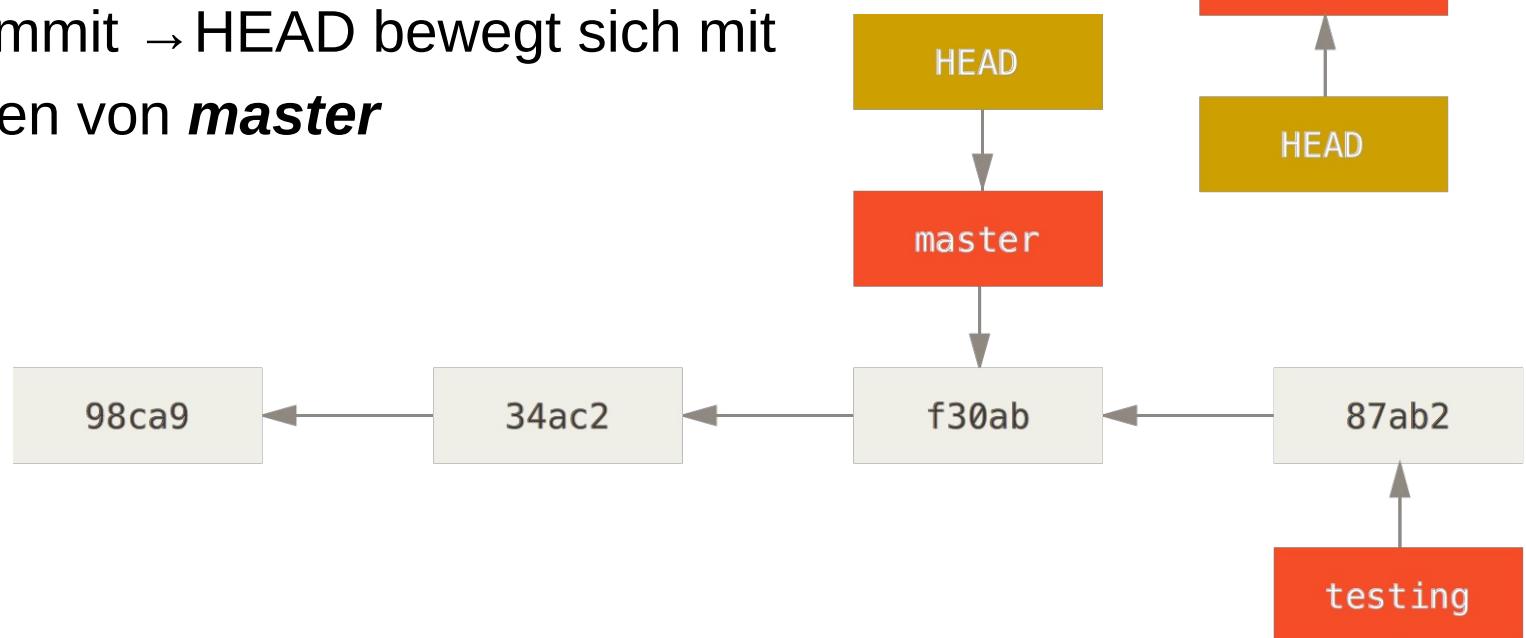
@soychotic

Every time I have a programming question and I rly need help, I post it on Reddit and then log into another account and reply to it with an obscenely incorrect answer. Ppl don't care about helping others but they LOVE correcting others. Works 100% of the time

Branches

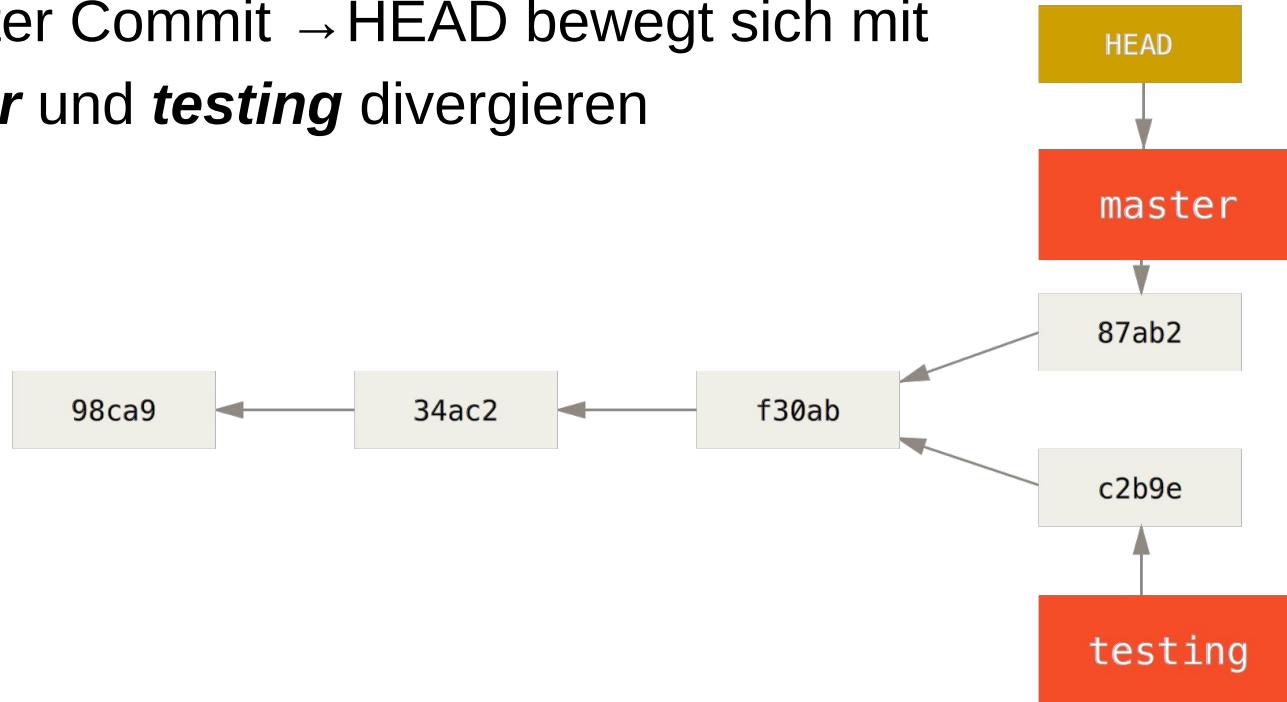


- Neuer Commit → HEAD bewegt sich mit
- Auschecken von **master**



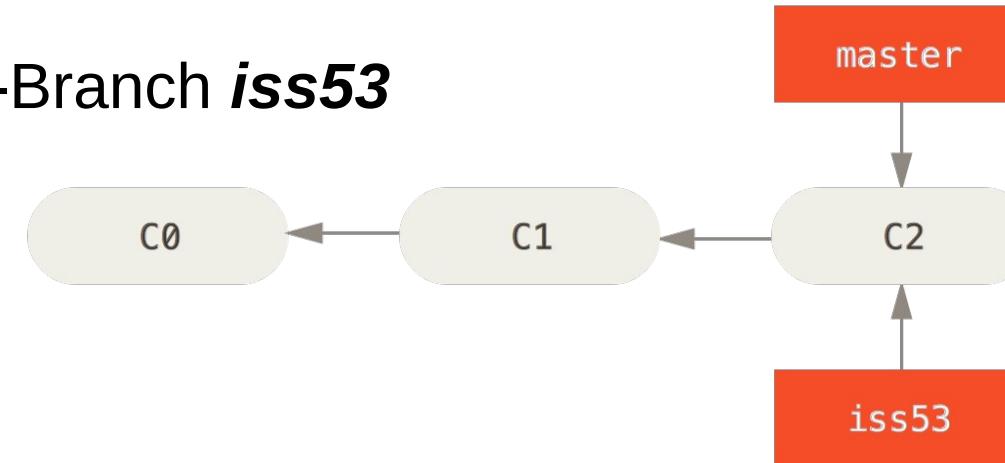
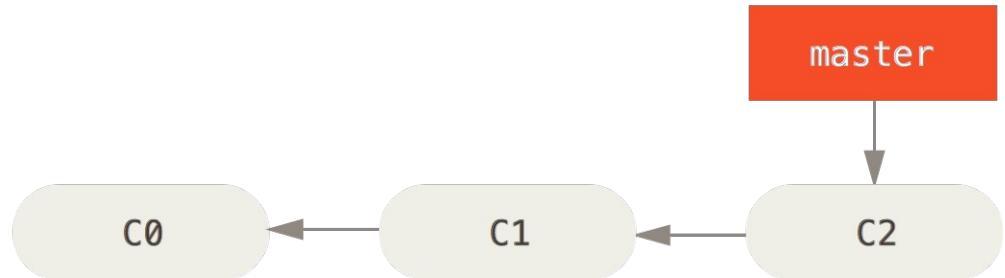
Branches

- Erneuter Commit → HEAD bewegt sich mit
- ***master*** und ***testing*** divergieren

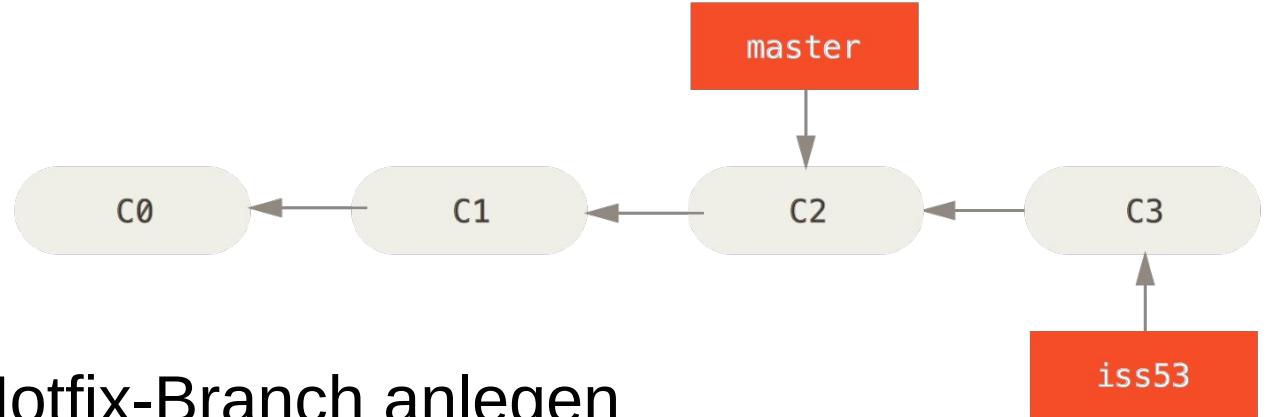


Branching

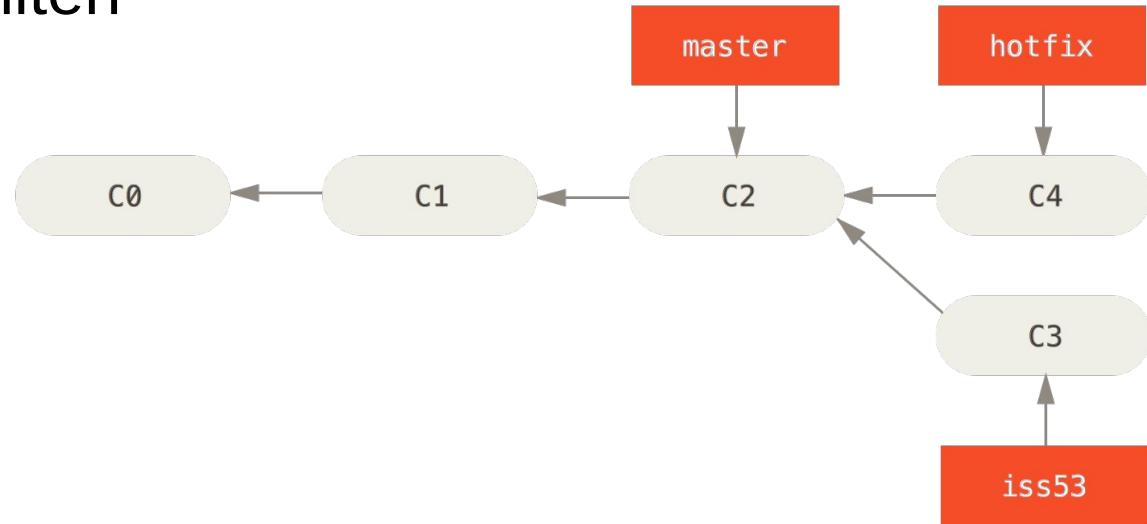
- Ausgangslage:
 - **master** und Feature-Branch **iss53**



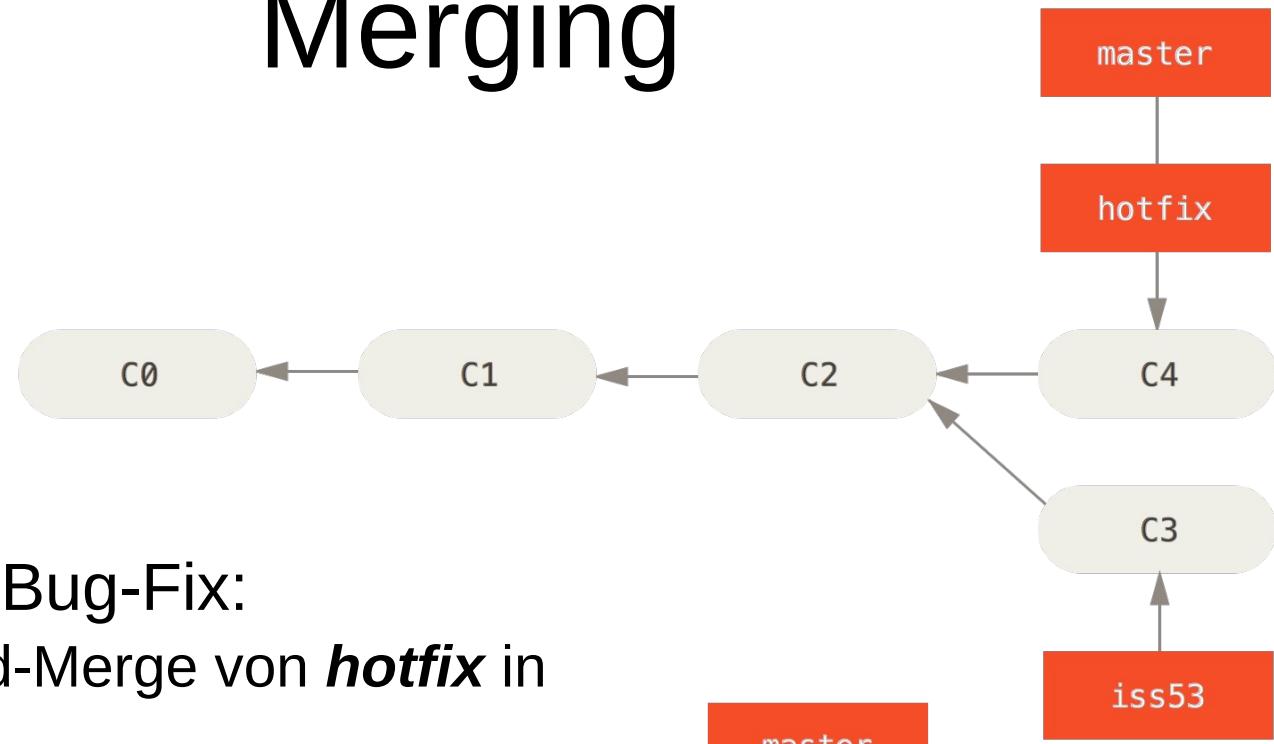
Branching



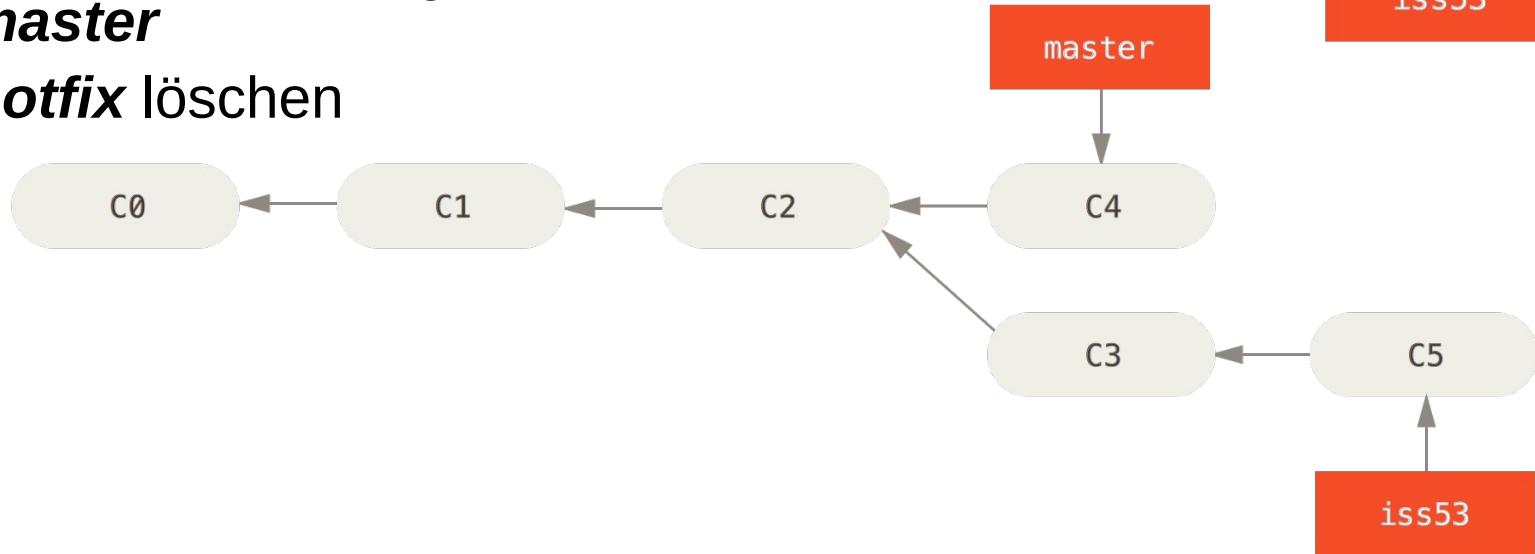
- Bug-Report → Hotfix-Branch anlegen
- Fix in **hotfix** commiten



Merging

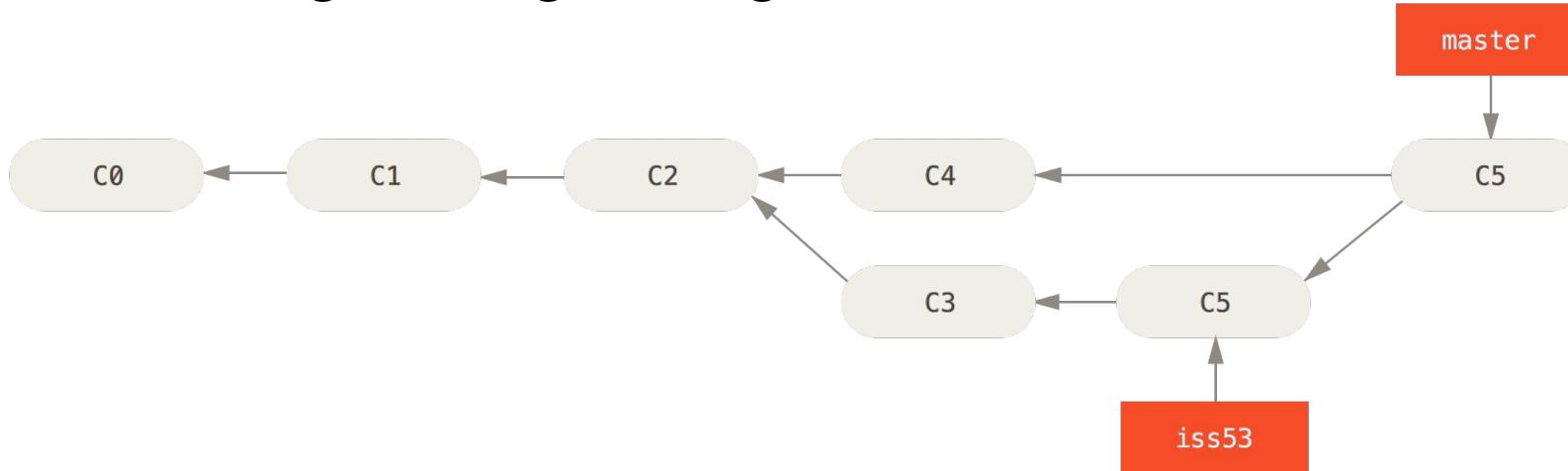
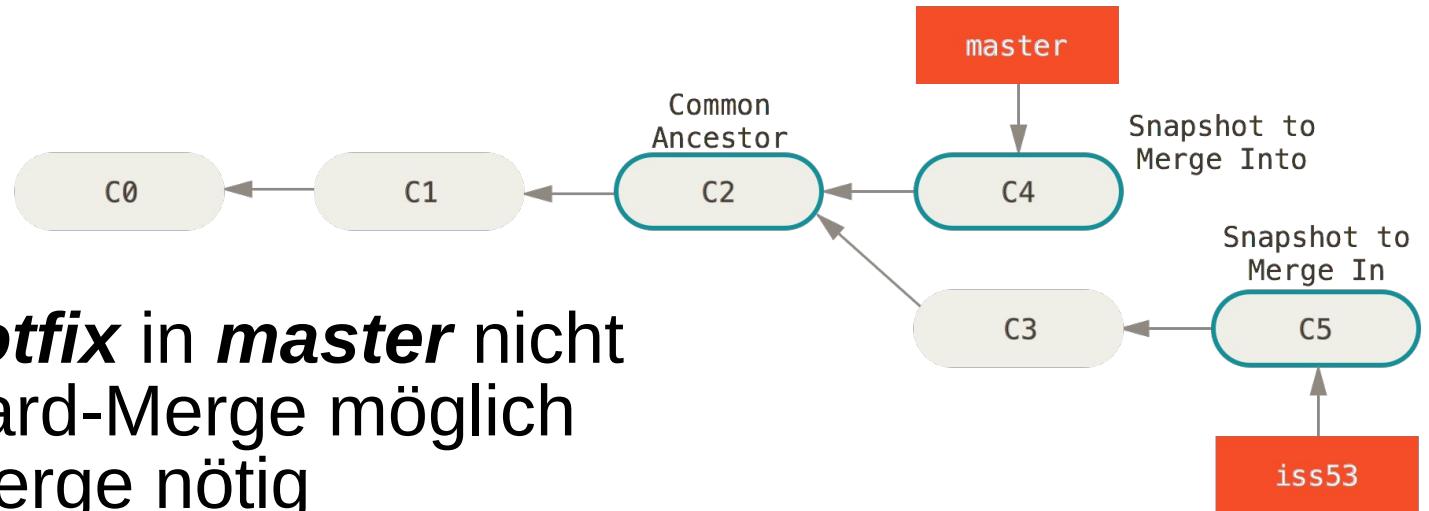


- Freigabe des Bug-Fix:
 - Fast-Forward-Merge von **hotfix** in **master**
 - **hotfix** löschen

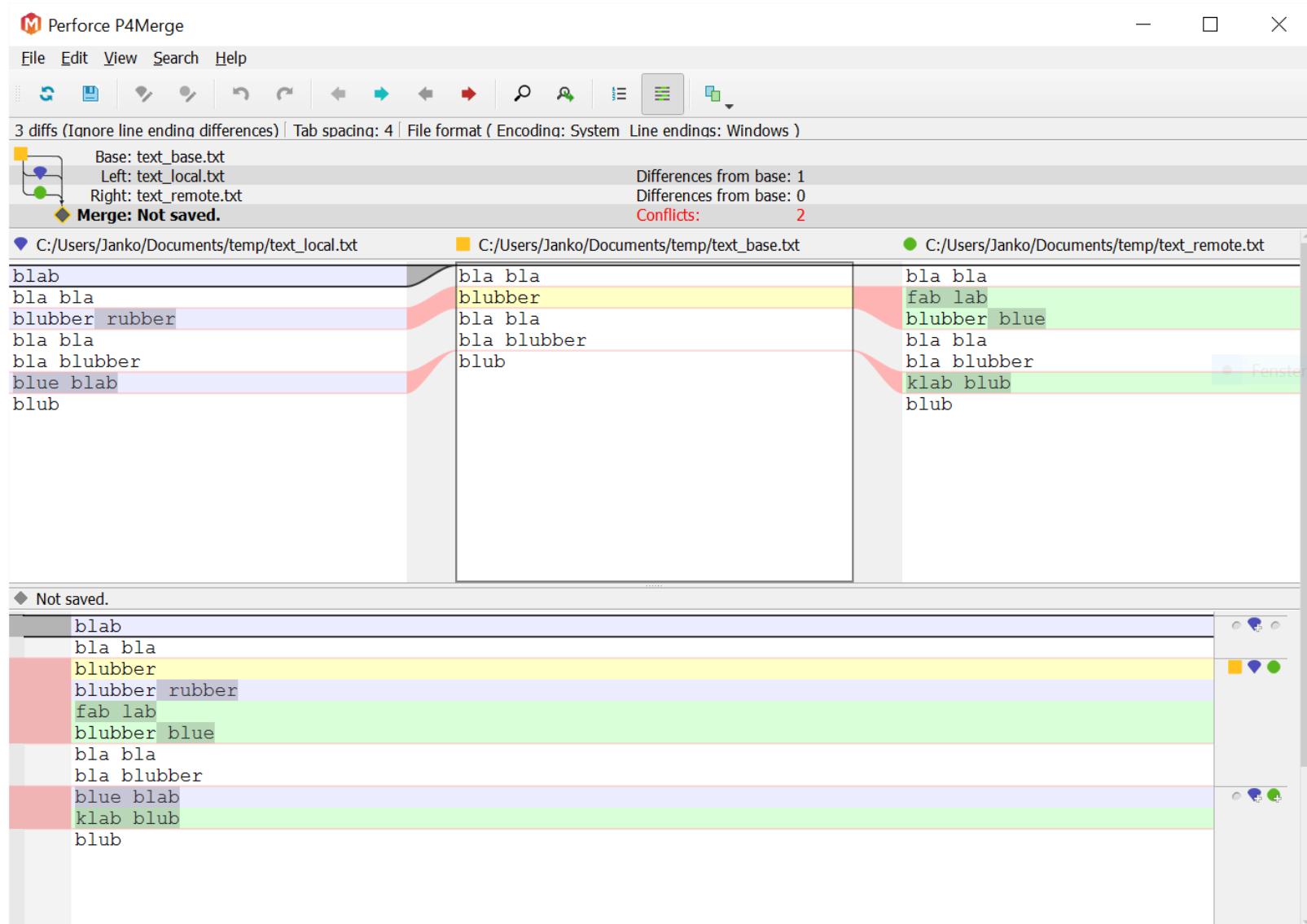


Branching/Merging

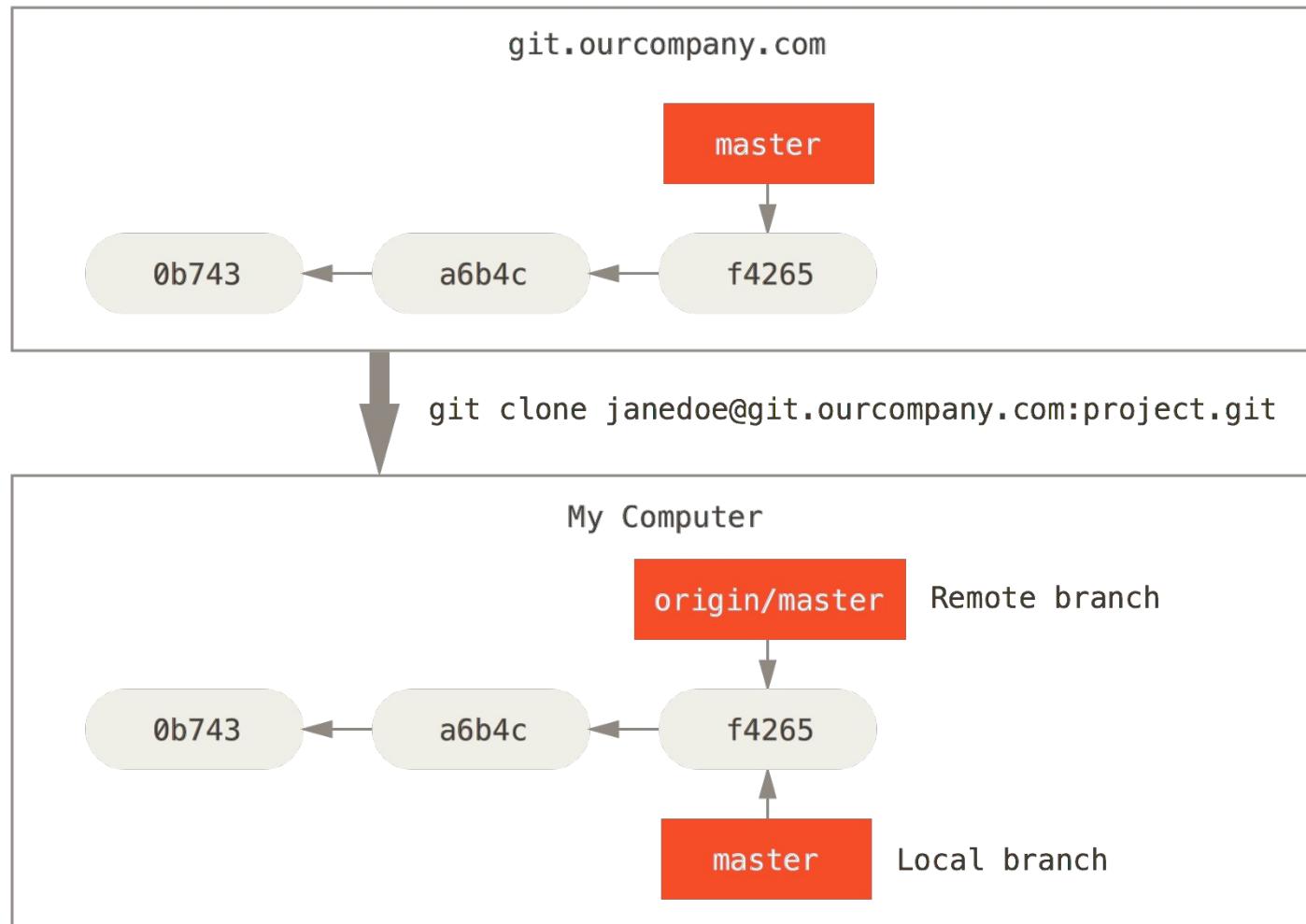
- Merge von **hotfix** in **master** nicht als Fast-Foward-Merge möglich
→ 3-Wege-Merge nötig



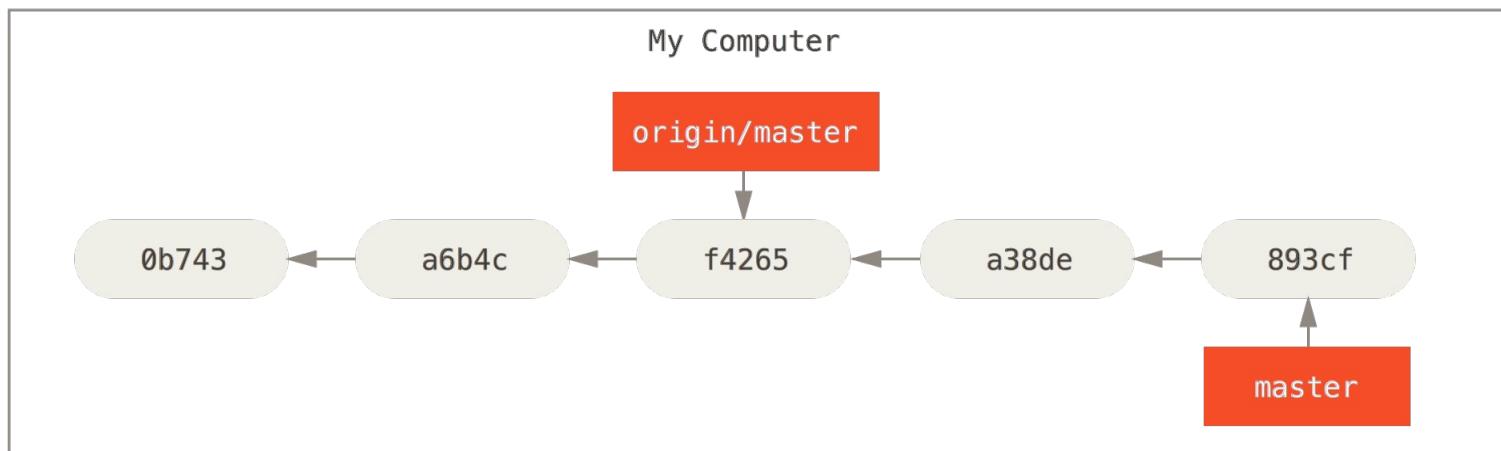
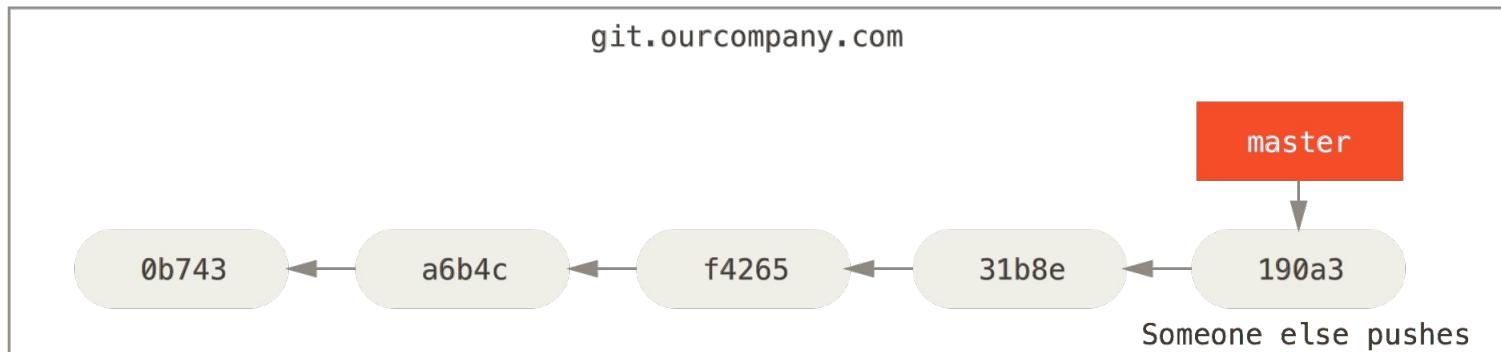
3-Wege-Merge



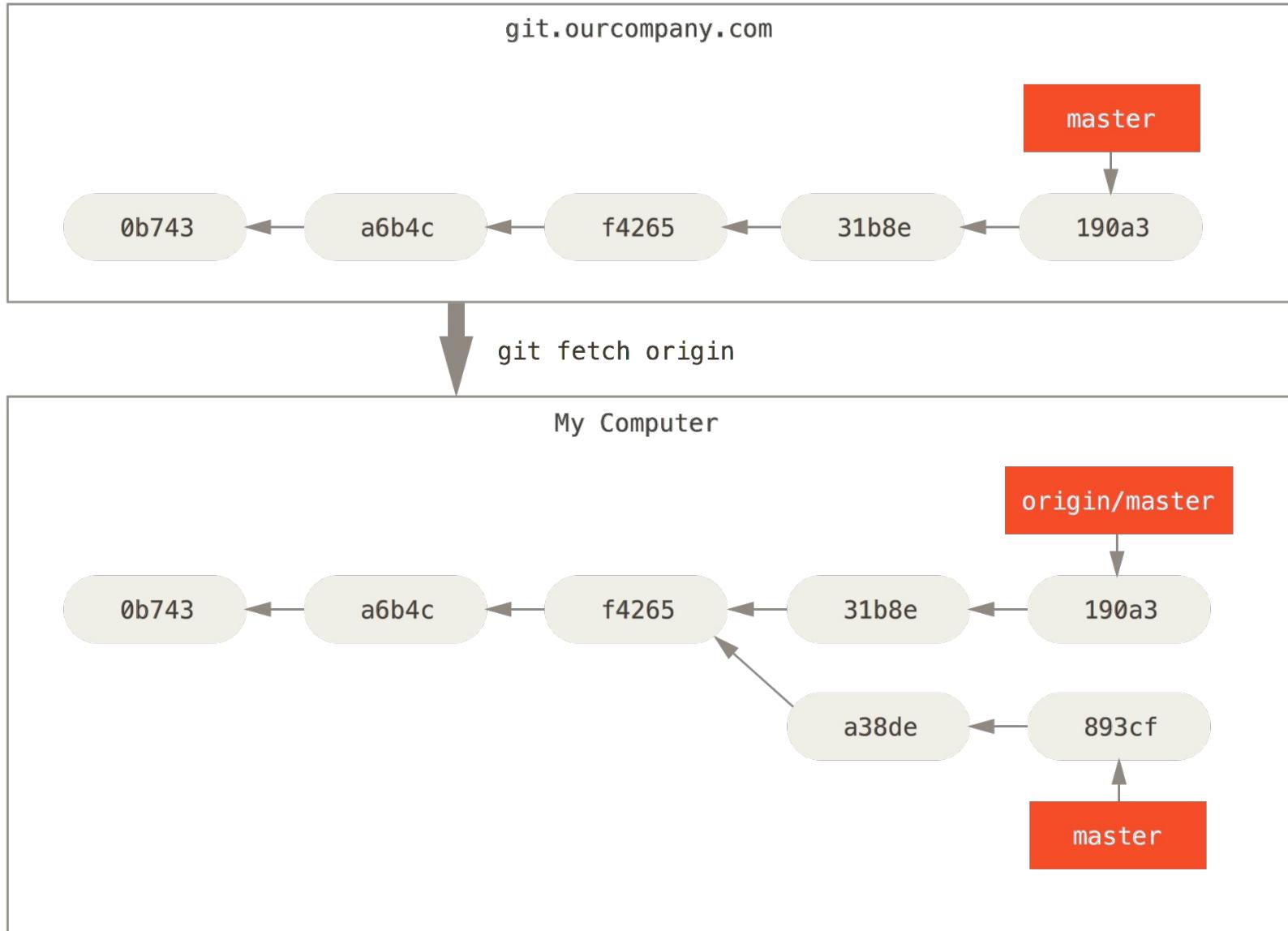
Remote Repository / Tracking-Refs



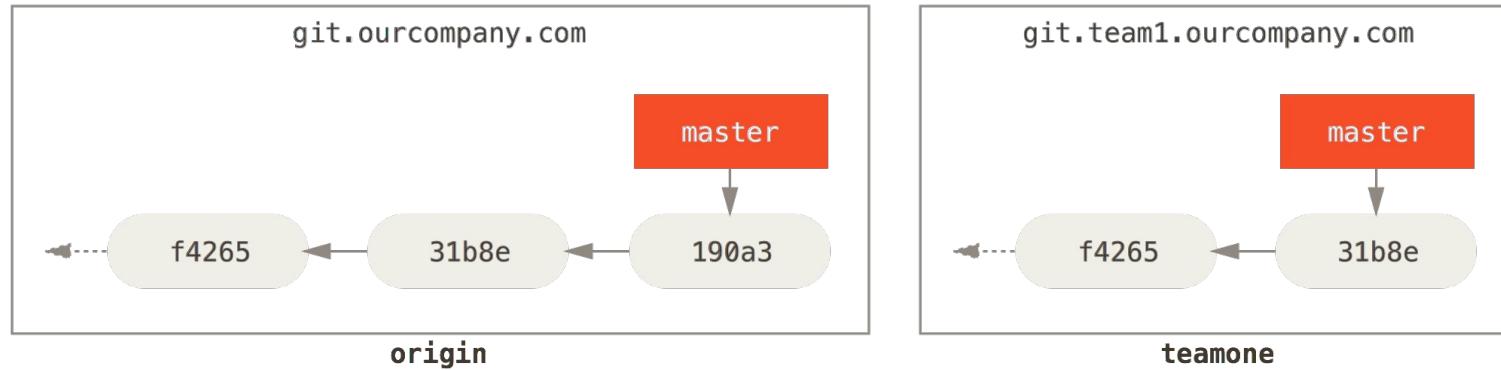
Remote Repository



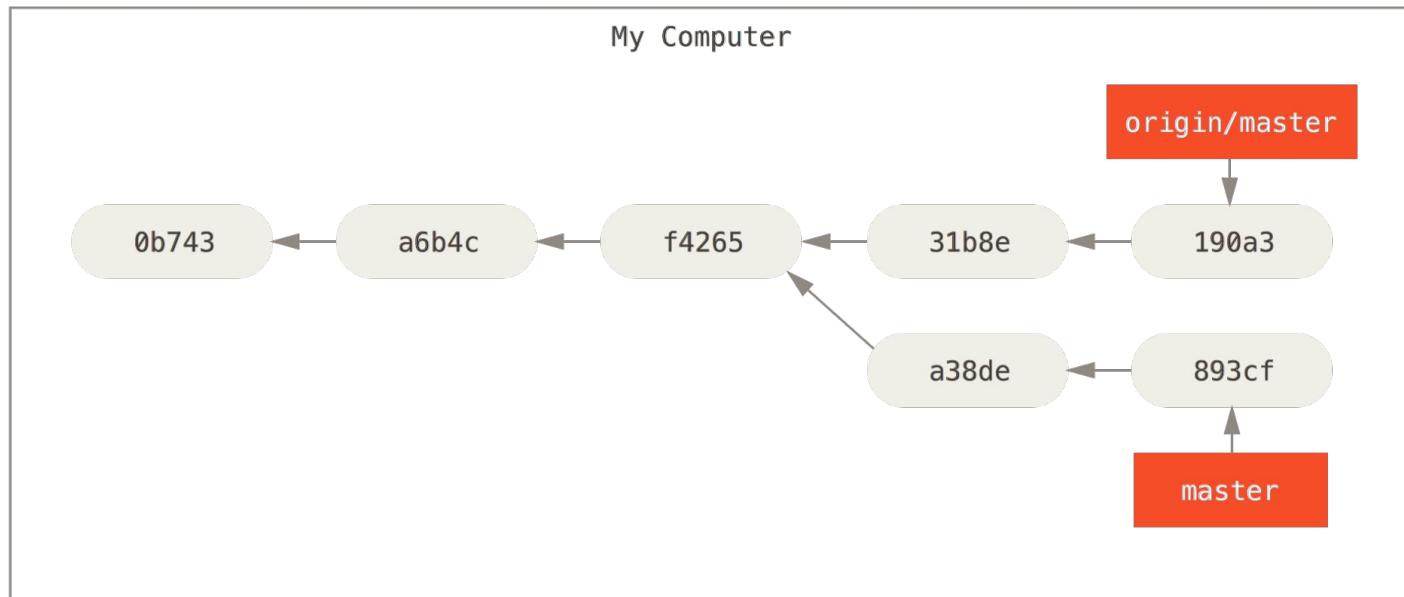
Remote Repository



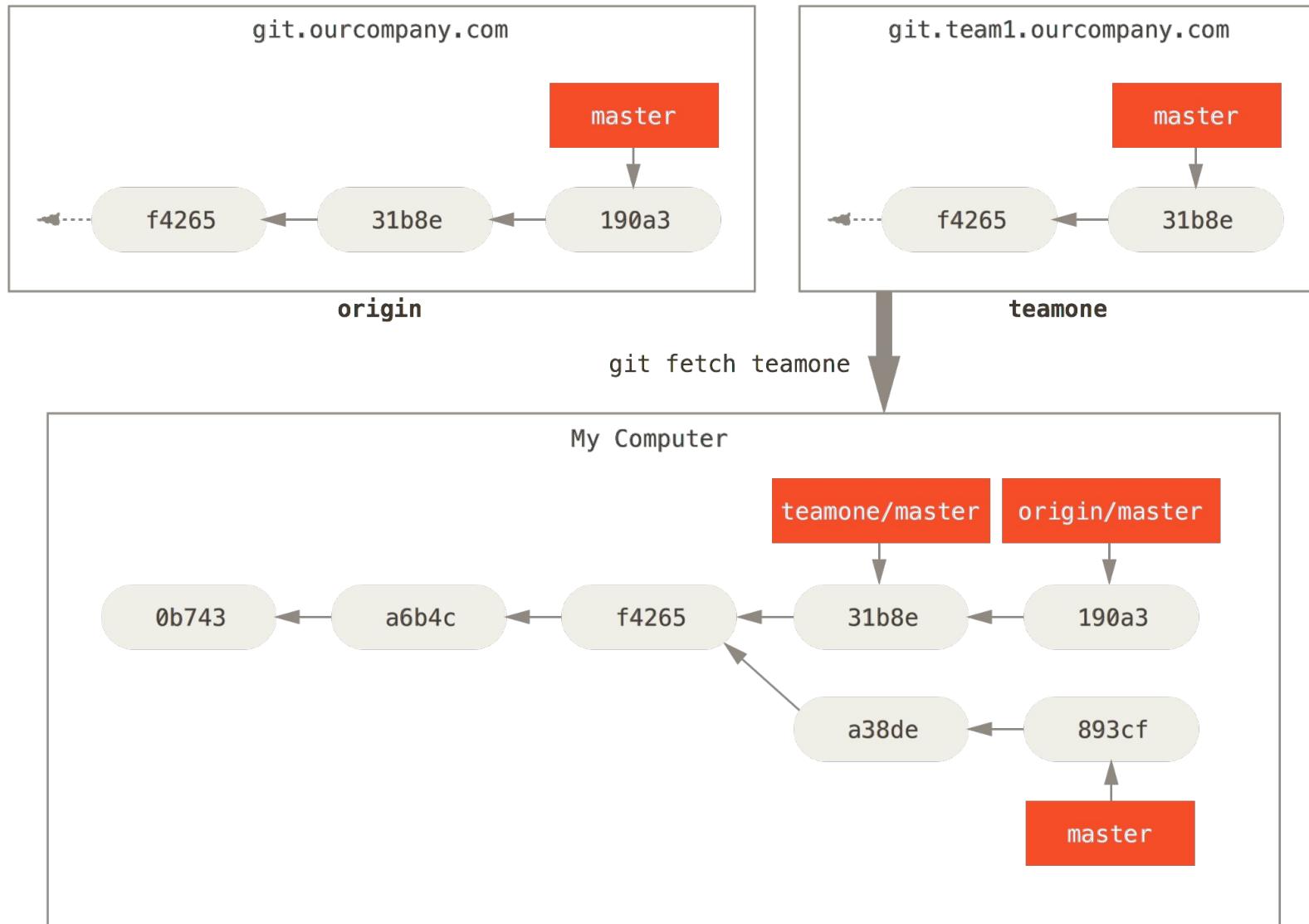
Ein weiteres Remote Repository



```
git remote add teamone git://git.team1.ourcompany.com
```



Ein weiteres Remote Repository

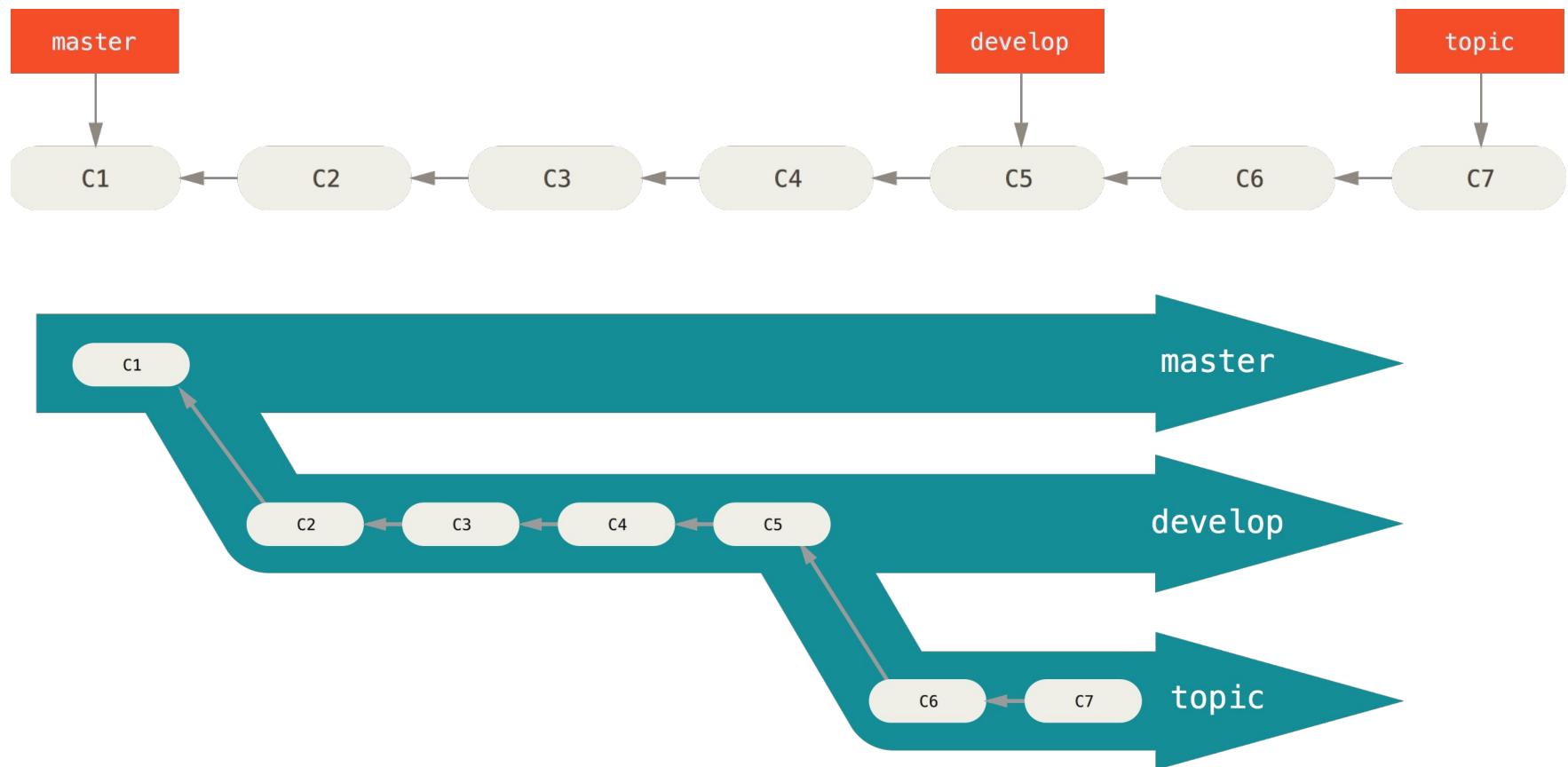


Branching-Workflows

- Master-Workflow
- Master/Develop-Workflow
- Kombinieren mit Feature/Topic Branches
- Gitflow
- Fork-Workflow, GitHub-Flow, GitLab-Flow, ...

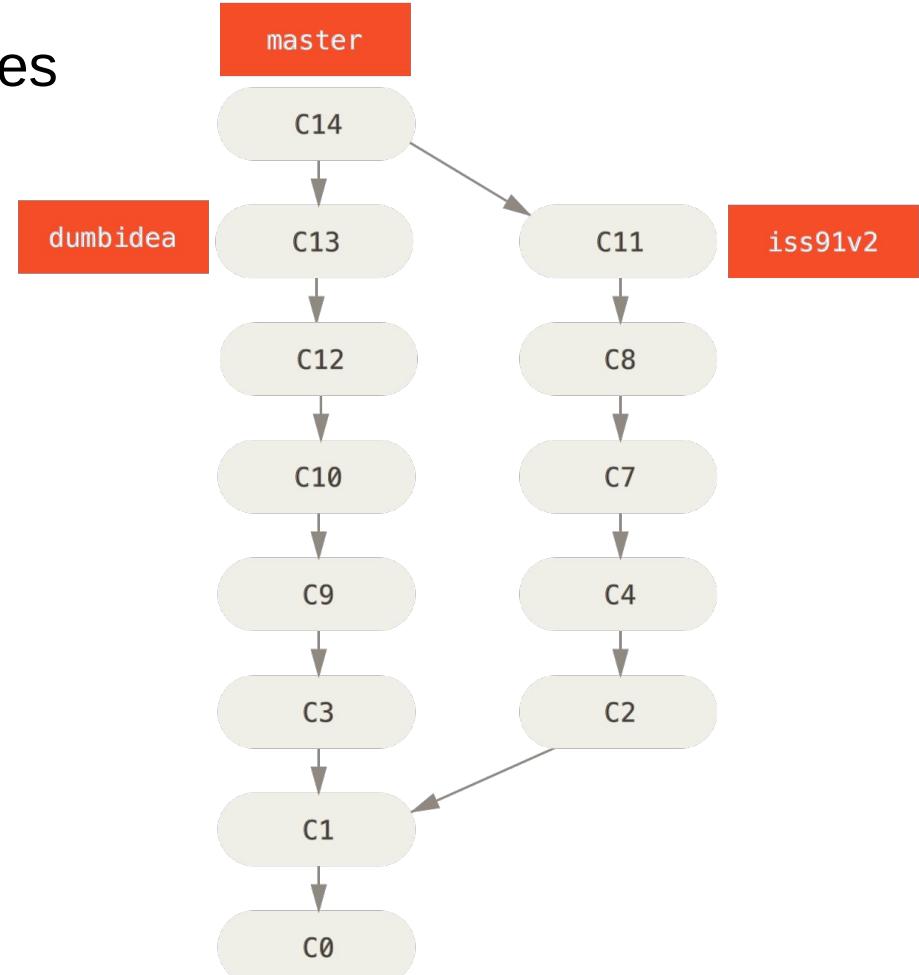
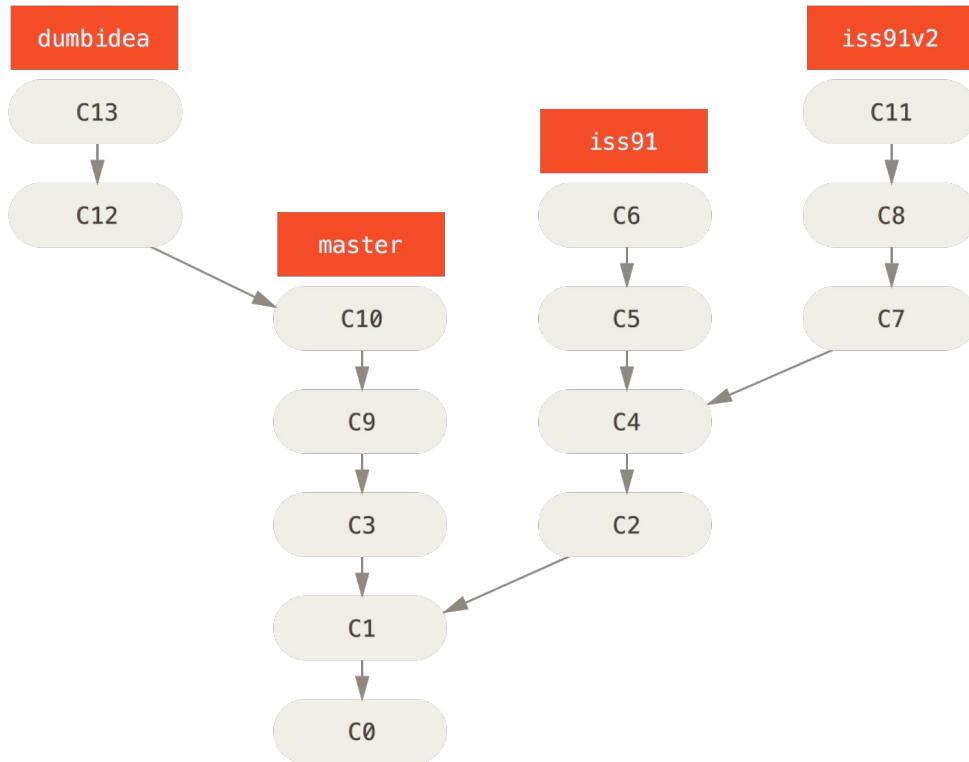
Einfacher Workflow

- Master-Workflow



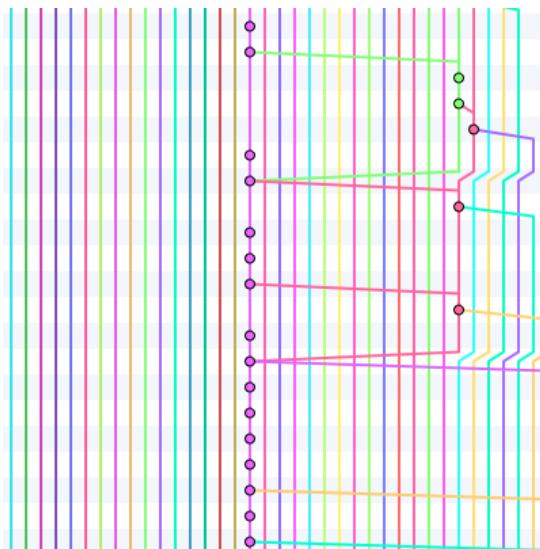
Einfacher Workflow

- Workflow mit vielen Feature-Banches

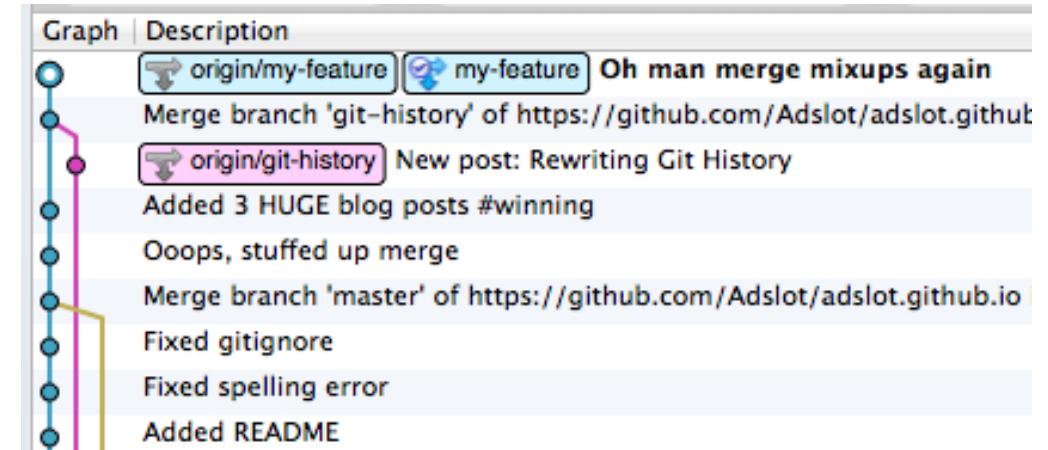


Rewrite History

- Motivation:
 - Bsp. von <http://engineering.adslot.com/2014/03/28/rewriting-git-history/>
 - „Branch-Verhau“ angeblich mit nur 3 Entwicklern
 - Manchmal „... mehr Commits als geänderte Quellcode-Zeilen“



styled dates change
Merge branch 'adver'
updated campaign st
Merge branch 'camp:
Merge branch 'camp:
first cut styling
Merge branch 'camp:
Merge branch 'camp:
advertiser change-re
advertiser can respo
Merge branch 'camp:
Merge branch 'camp:
Initial advertiser cont
Merge branch 'camp:
cleaned up code
remove overbook res
added sponsorship d
sponsorship availab
Merge branch 'maste
style validation error
Merge branch 'maste



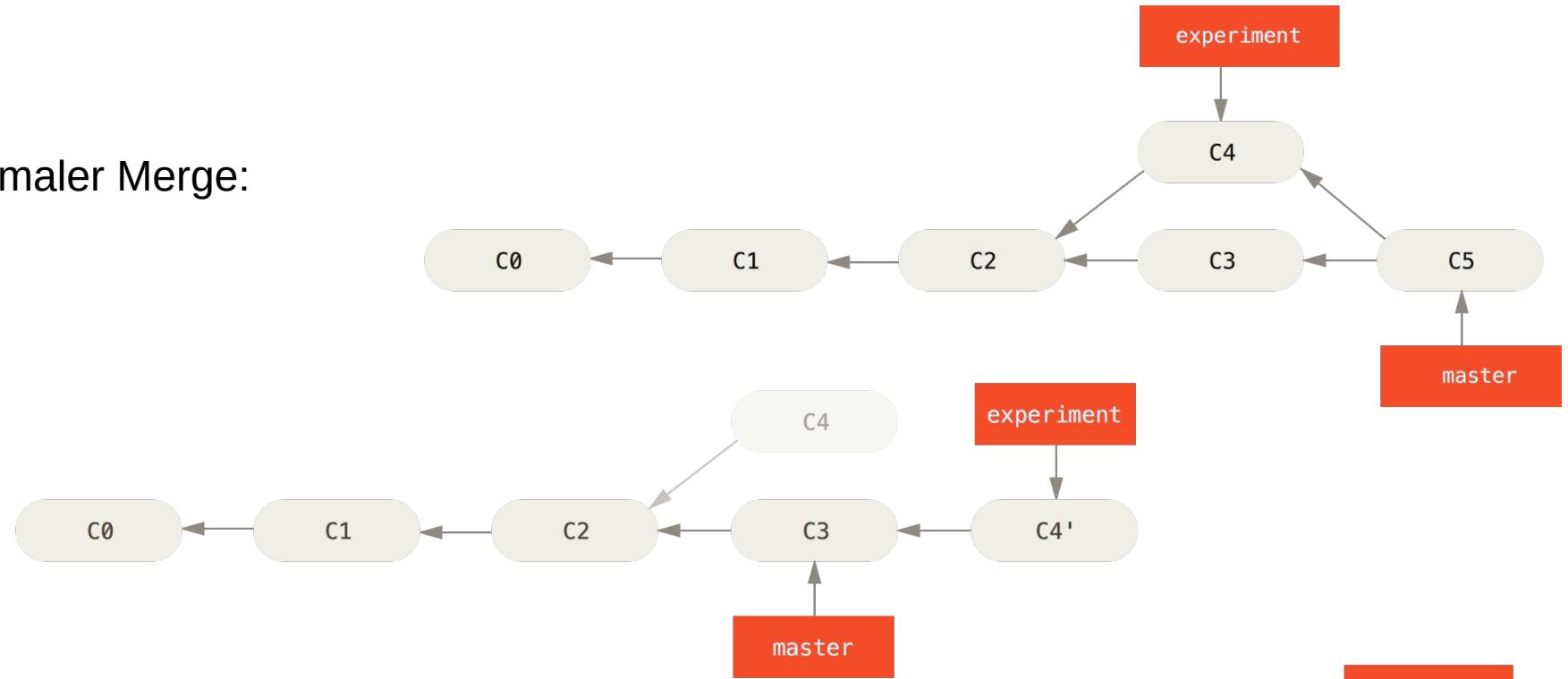
Rewrite History

- Ziel:
 - Mit Hilfe von *rebase*, *amend*, *reset*, *cherry-pick* ... (*reflog*)

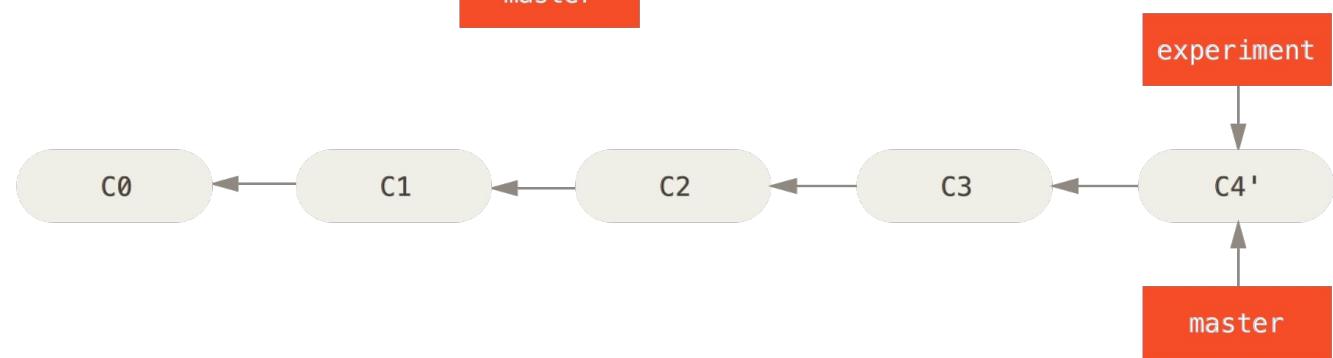


Einfacher Rebase

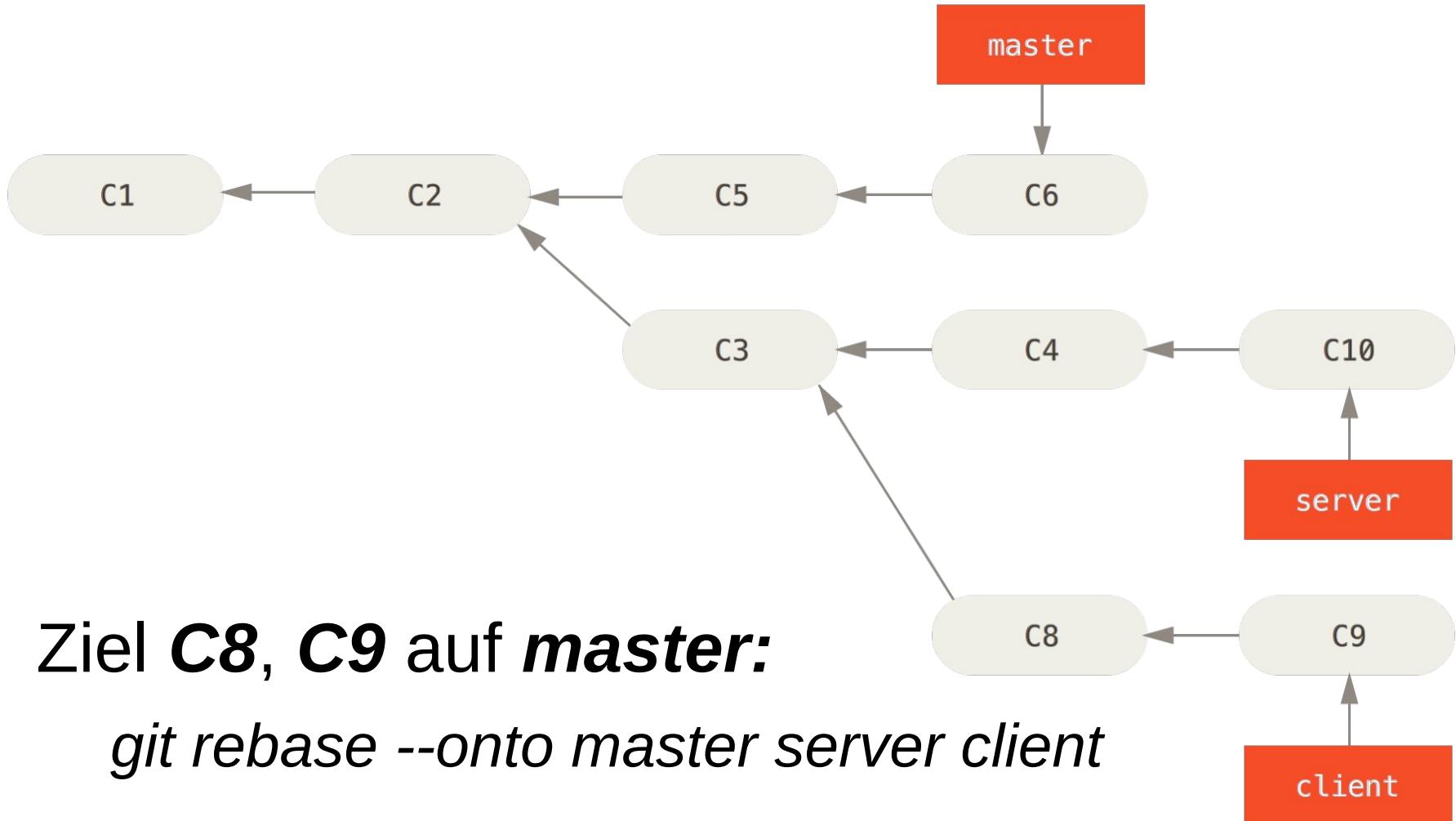
Normaler Merge:



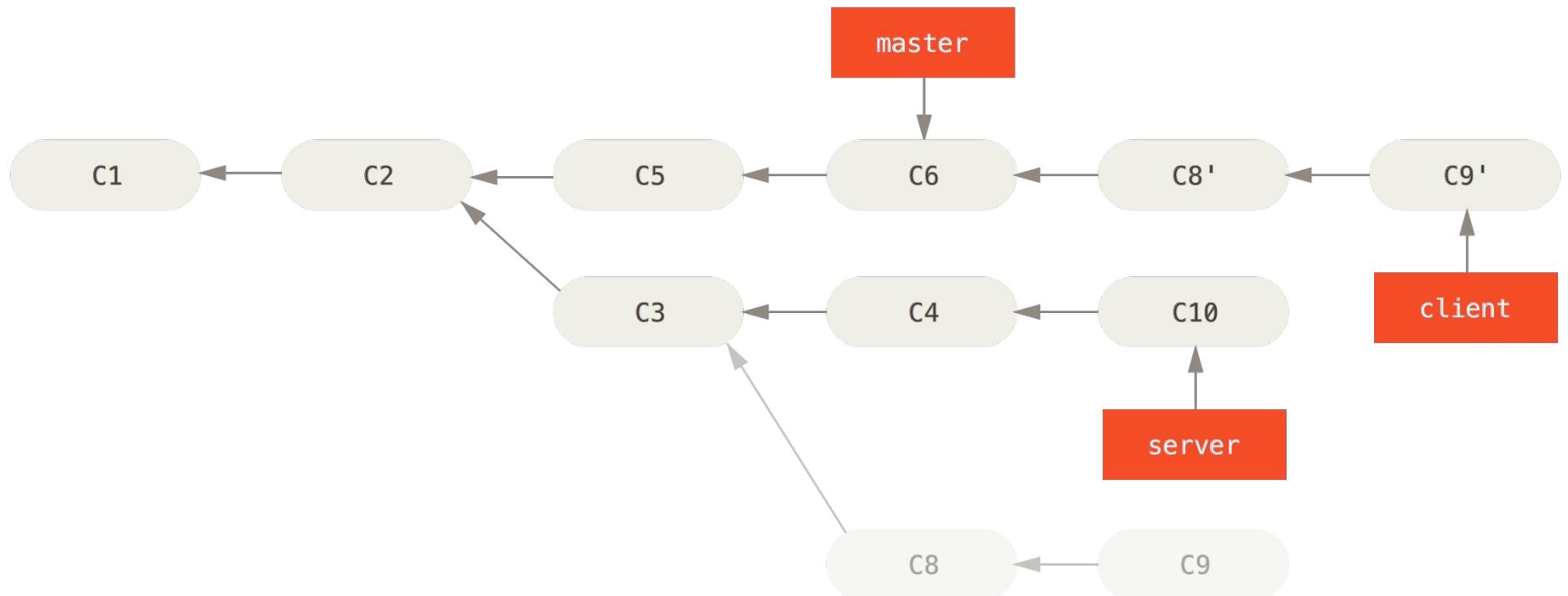
Rebase:



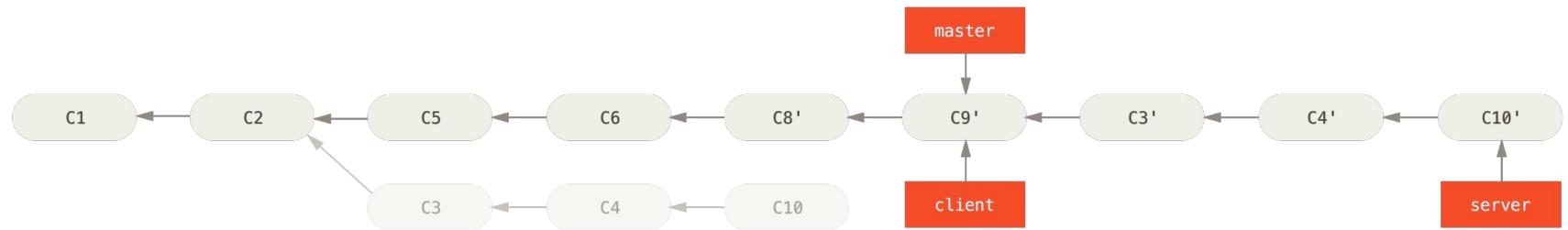
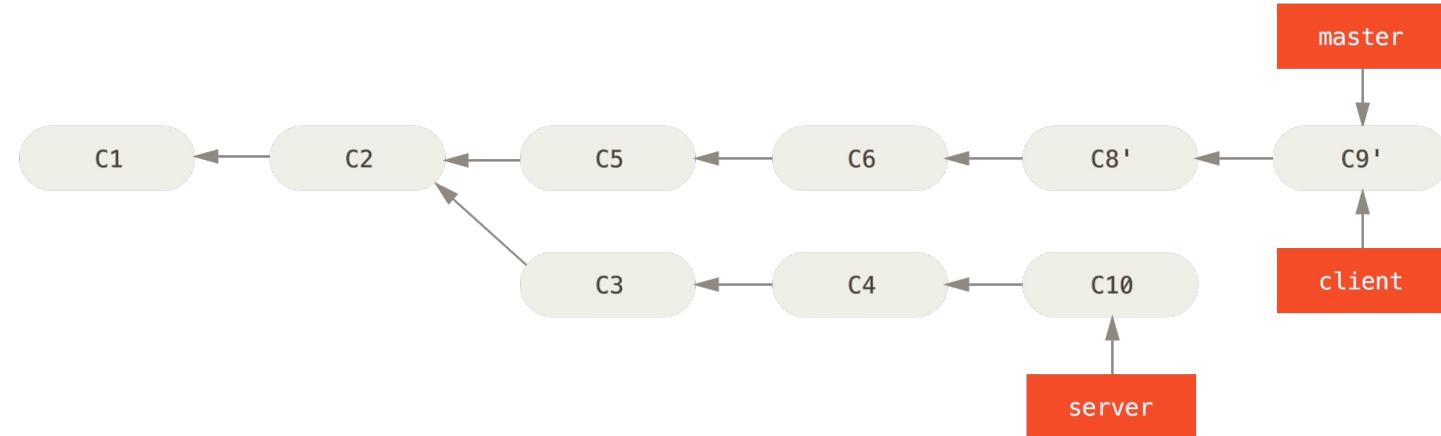
Komplexer Rebase



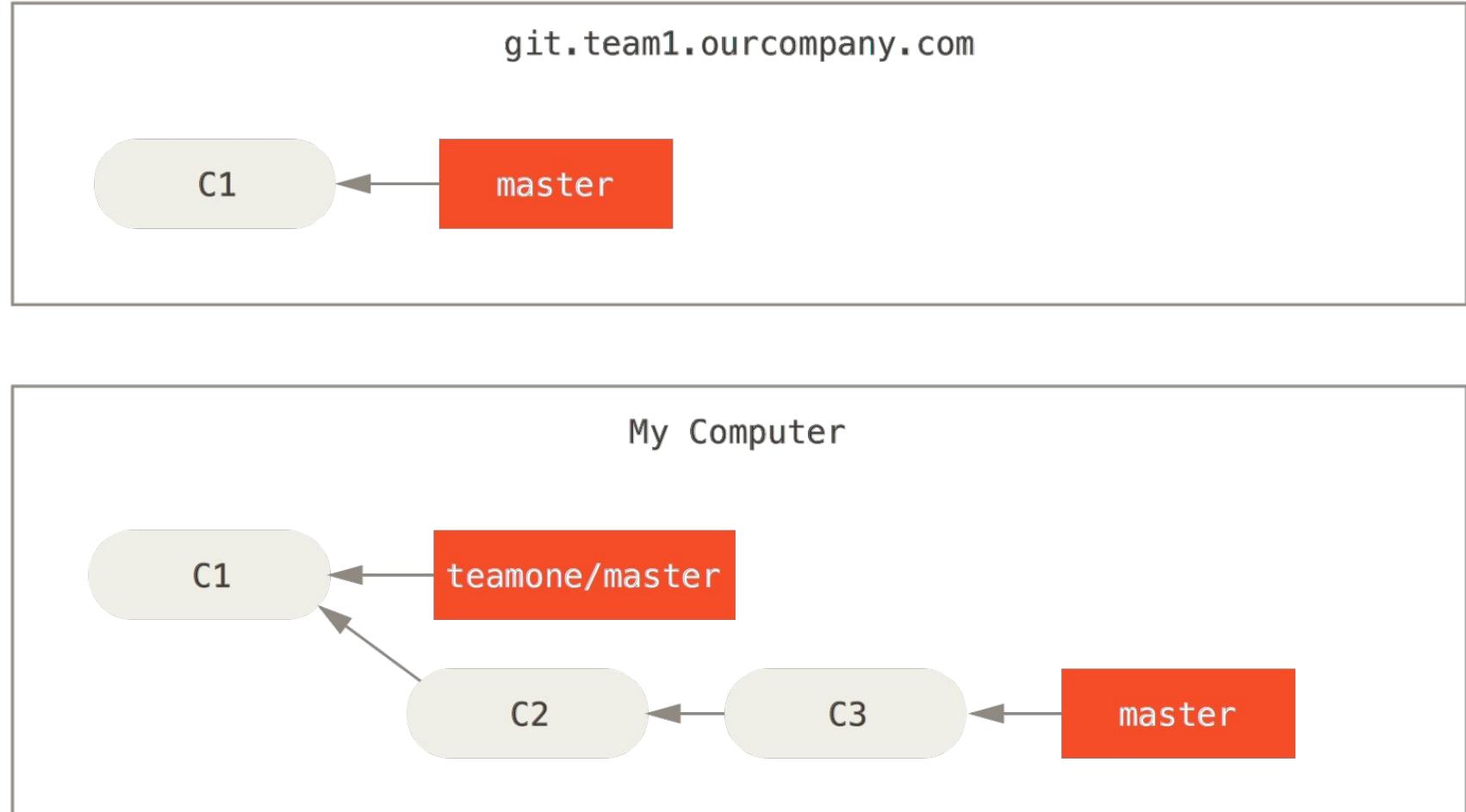
Rebase *client* auf *master*



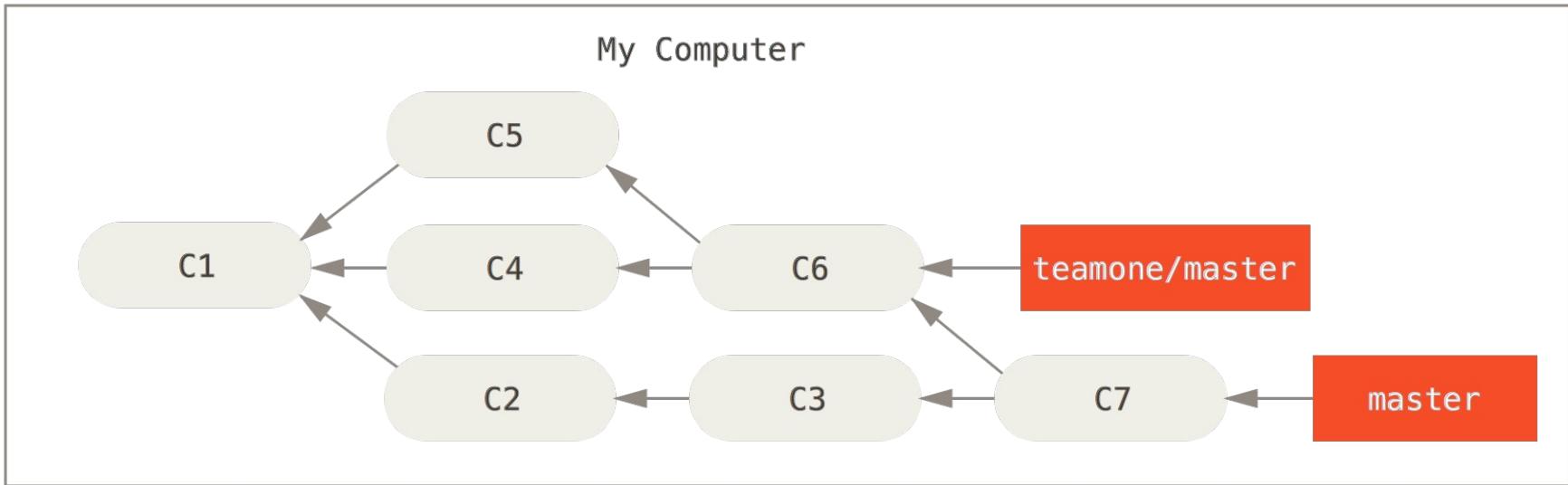
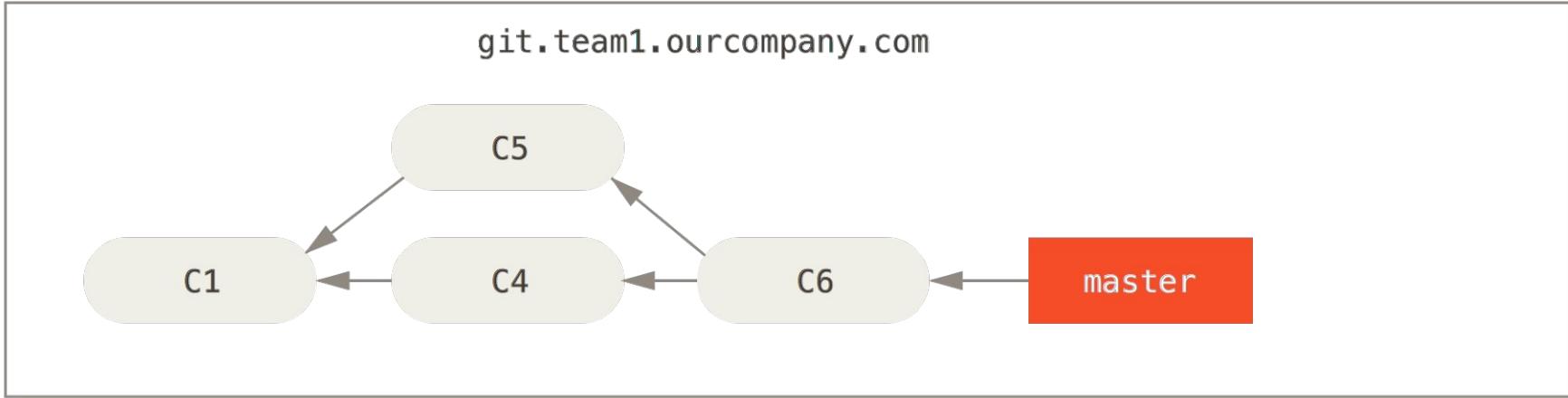
Rebase *server* auf *master*



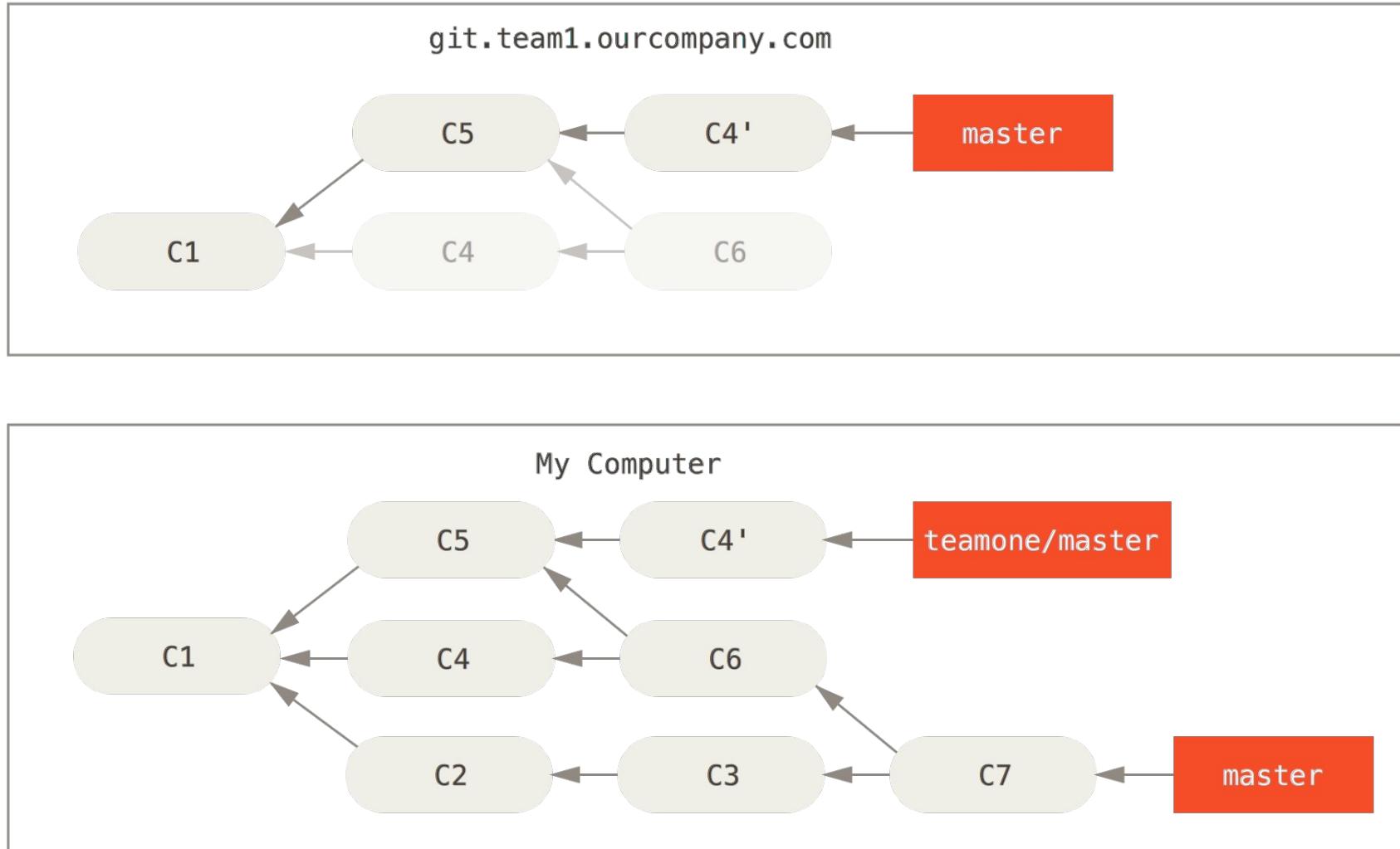
Rebase Gefahren



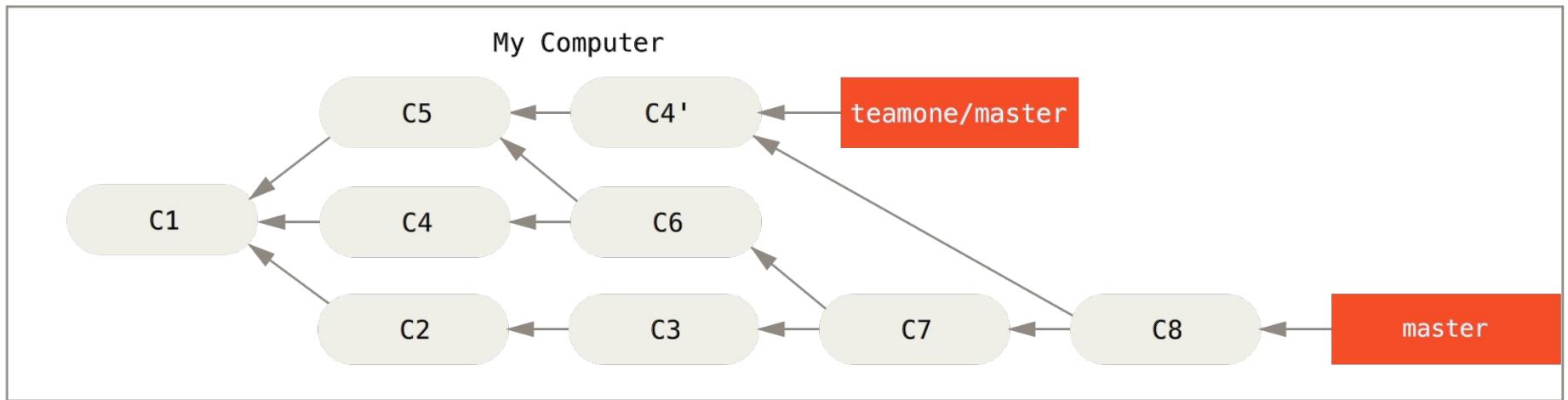
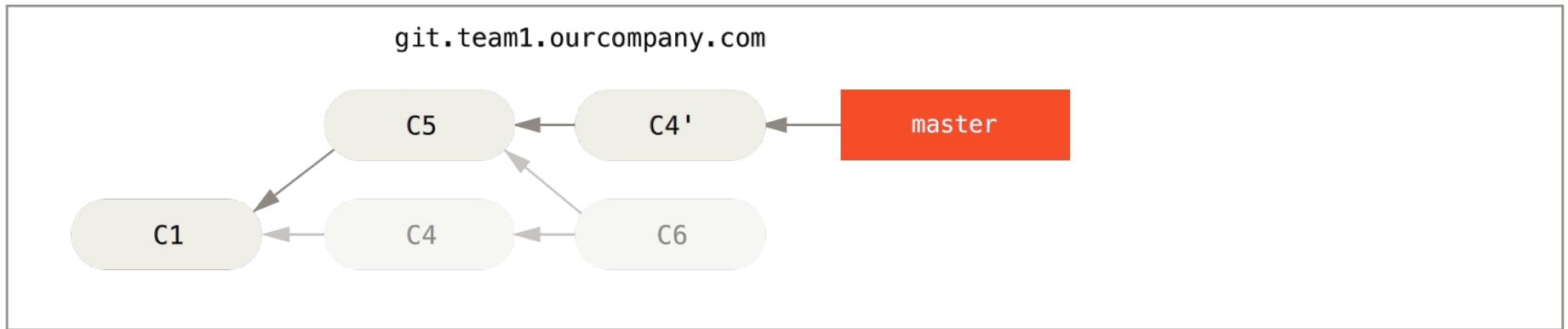
Rebase Gefahren



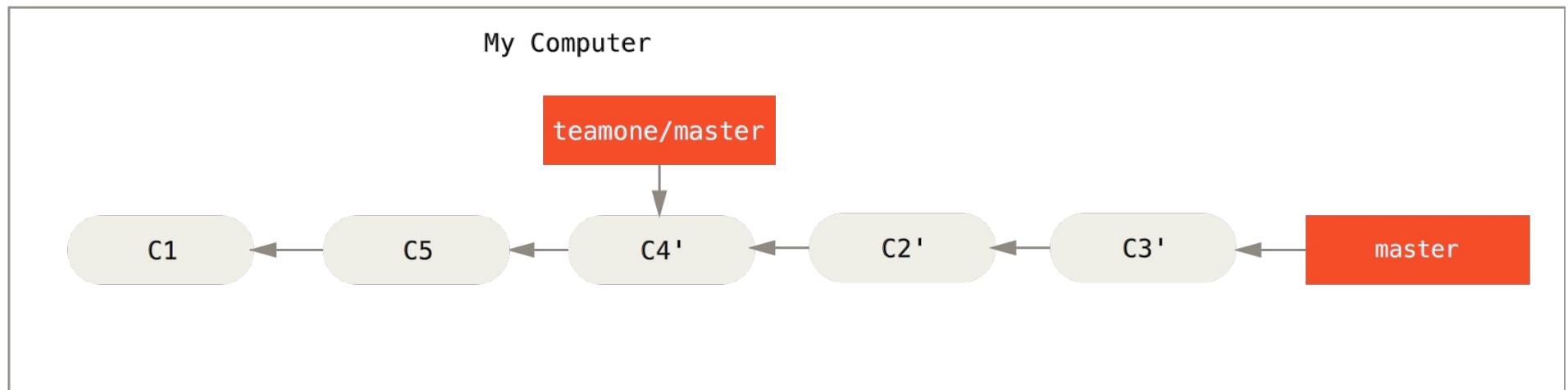
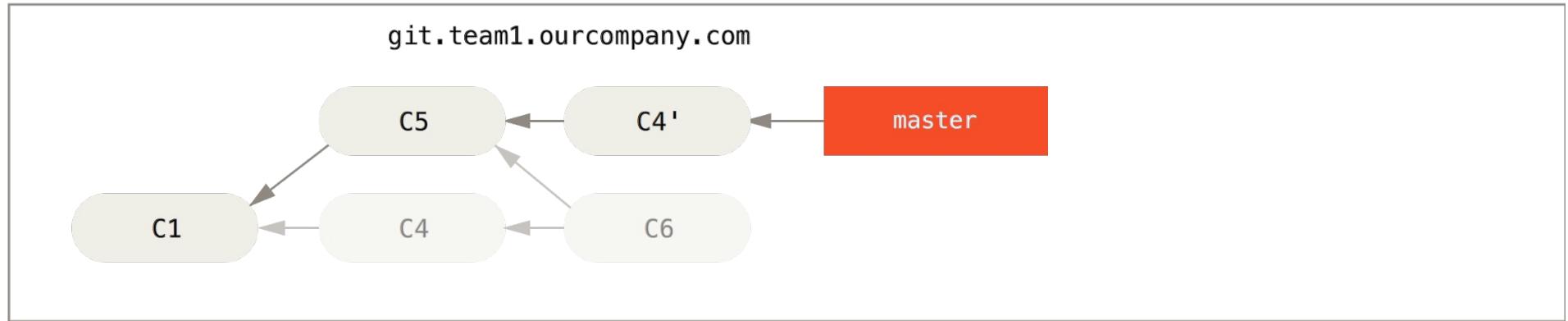
Rebase Gefahren



Rebase Gefahren



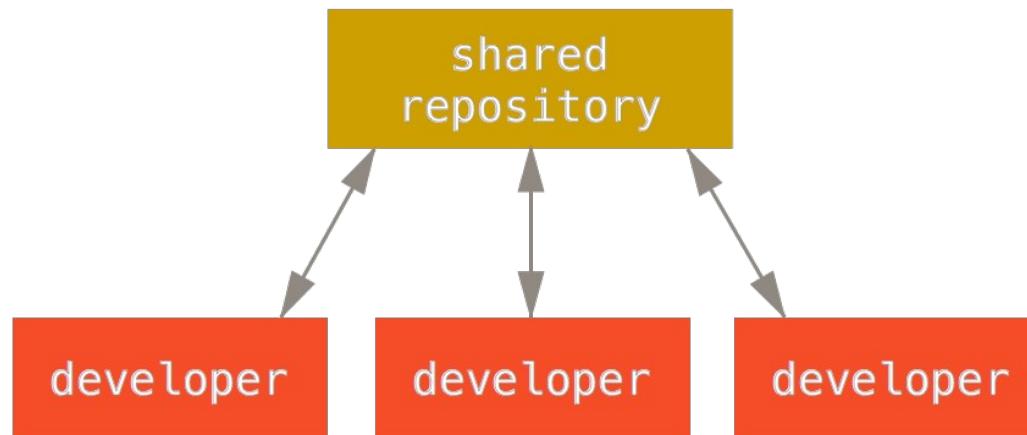
Rebase – Rebase statt Merge lokal



Team-Workflow/Distributed-Workflow

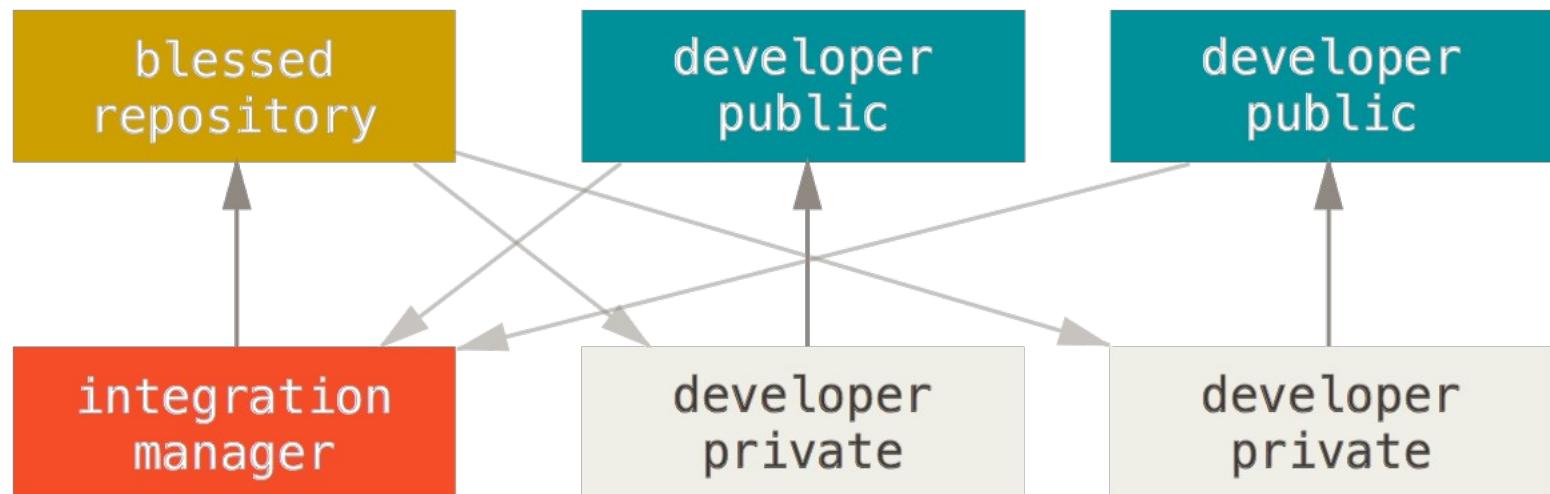
- Bisherige Betrachtungsebene:
 - lokaler Workflow
 - Gegenstand ist eine möglichst günstigste Organisation der Branches
 - Organisation des „Persönlichen VCS“
- Jetzt:
 - Workflow auf der Ebene des Teams
 - Kollaborationsaspekt des VCS
 - Herkömmliche zentralisierte VCS (CVCS) legen diesen von vornherein fest
 - Mit verteilten VCS (DVCS) lassen sich hingegen verschiedene Varianten realisieren, da jedes einzelne Repository sowohl als „Client“ wie auch als „Server“ fungieren kann
- Zentralisierter Workflow, Fork-Workflow, ...

Zentralisierter Workflow



- Ein zentrales Repository auf dem Server über das der gemeinsame Code ausgetauscht wird und das den gemeinsamen aktuellen Code-Stand definiert
- Gewohnter Workflow (SVN, ...)
- „Parallele“ Änderungen müssen erst im lokalen Repository gemergt und dann gepusht werden
- Zentrales Repository ist meistens ein **Bare**-Repository, d.h. es besitzt kein direkt zugeordnetes Arbeitsverzeichnis:
 - mkdir ProjectName.git
 - cd ProjectName.git
 - git --bare init

Integration-Manager/Fork Workflow

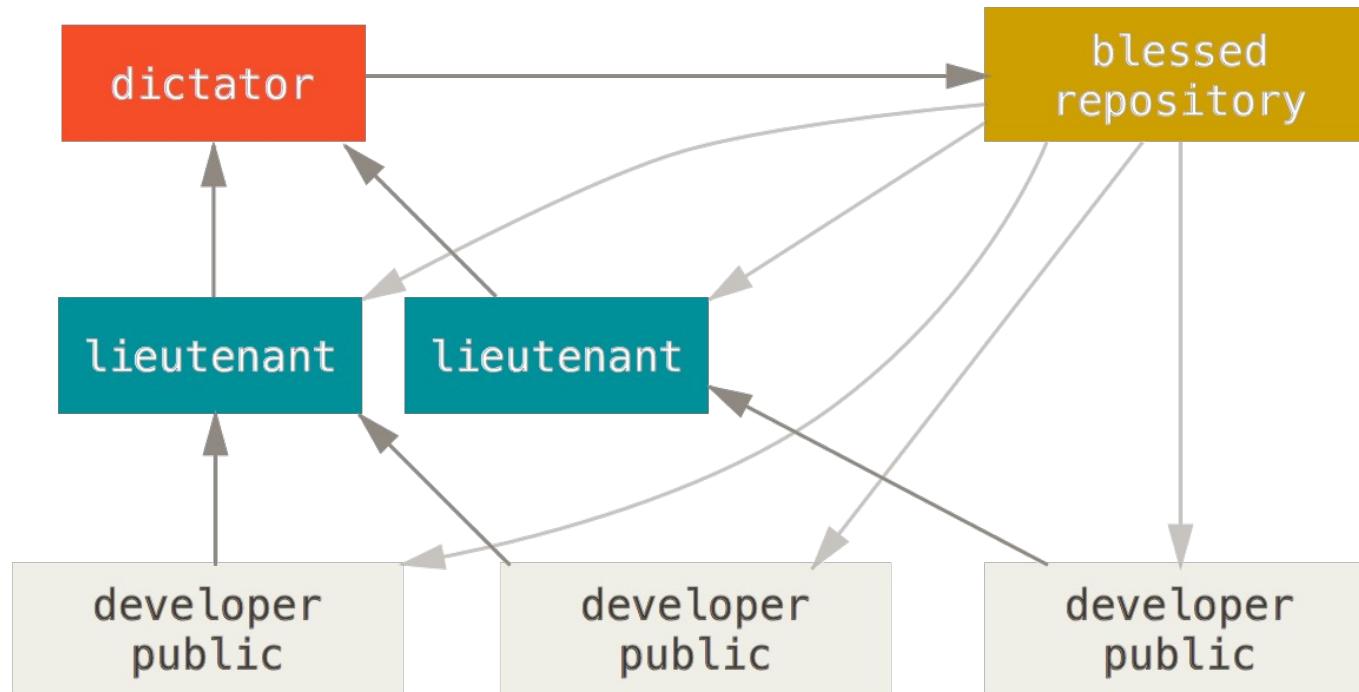


- Weit verbreitetes Vorgehen bei Open Source Projekten
- Gut unterstützt auf GitHub, BitBucket, ...
- Ein Projekt hat ein allgemein anerkanntes öffentliches Repository - **Blessed Repository** - auf das nur der Maintainer bzw. die Maintainer-Gruppe Schreibrechte besitzt
- Entwickler klonst dieses Repository in ein eigenes, privates Repository auf das nur er zugreift und mit dem er arbeitet – **Fork** des Projektes

Integration-Manager/Fork Workflow

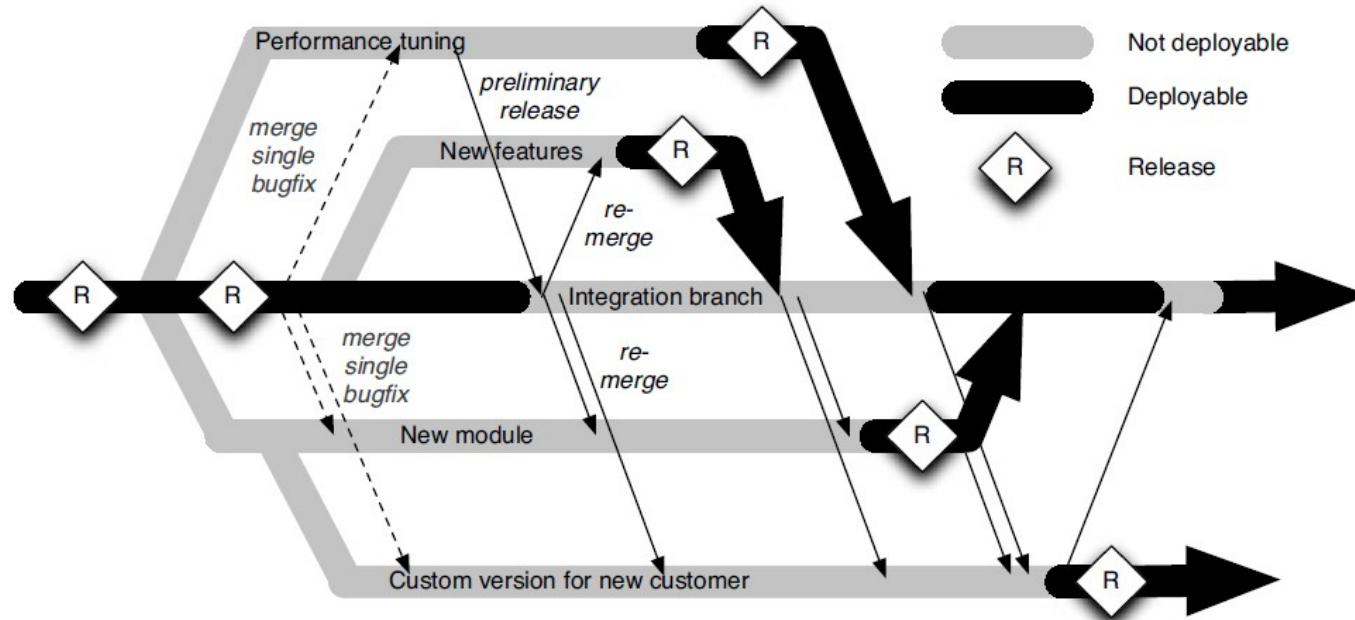
- Will ein Entwickler seine Änderungen in die Hauptlinie des Projektes bringen:
 - Anlegen eines eigenen öffentlichen Repository's
 - Pushen der zu integrierenden Änderungen
 - E-Mail an den Projekt-Maintainer bzw. die entsprechende Gruppe mit der Bitte, die Änderungen in die Hauptlinie zu integrieren → **Pull-Request**
 - Maintainer holt die Änderungen in sein privates Repository → Review, Diskussion, Refactoring → Integration
 - Anschließend pusht der Maintainer die Änderungen in das öffentliche Repository des Projektes
- Einen großen Vorteil stellt die gute Entkopplung dar
 - Jeder Entwickler kann in seinem privaten Fork unabhängig arbeiten
 - Der Maintainer kann jederzeit die Änderungswünsche der Entwickler übernehmen
 - Verwaltung der Zugriffs- bzw. Schreibrechte ist wesentlich vereinfacht

Diktator und Leutnants Workflow



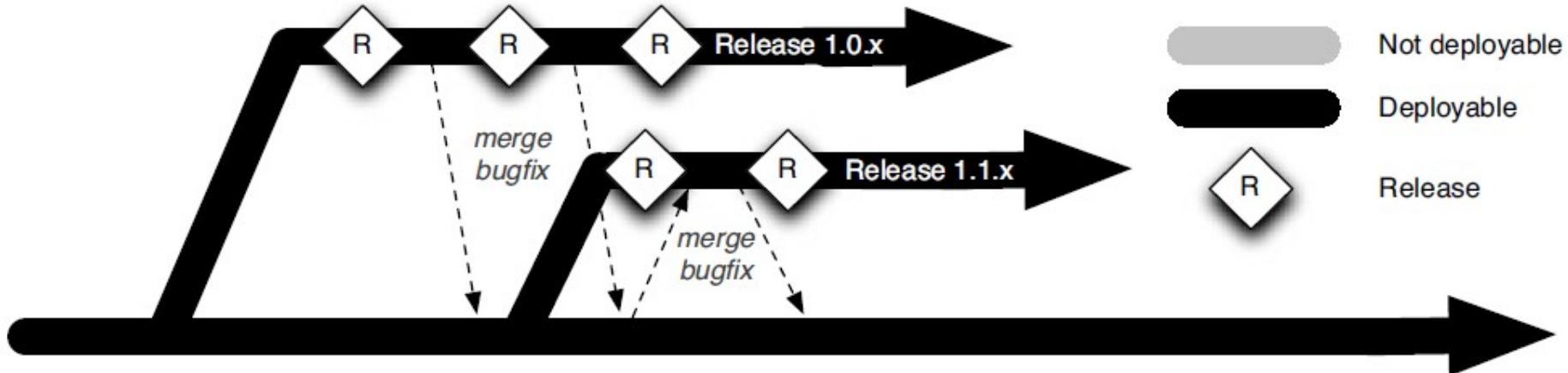
- Ausschließlich von großen Projekten bzw. Organisationsstrukturen eingesetzt, wie z.B. bei der Linux-Kernel-Entwicklung
- Zahlreiche Integration-Manager arbeiten einem zentralen „wohlwollenden Diktator“ („benevolent Dictator“, z.B. L. Torvalds) zu
- Integration-Manager sind für Teilbereiche des Projektes bzw. des Codes zuständig, z.B. Kernel-Subsystem

Branching- & Release-Strategie



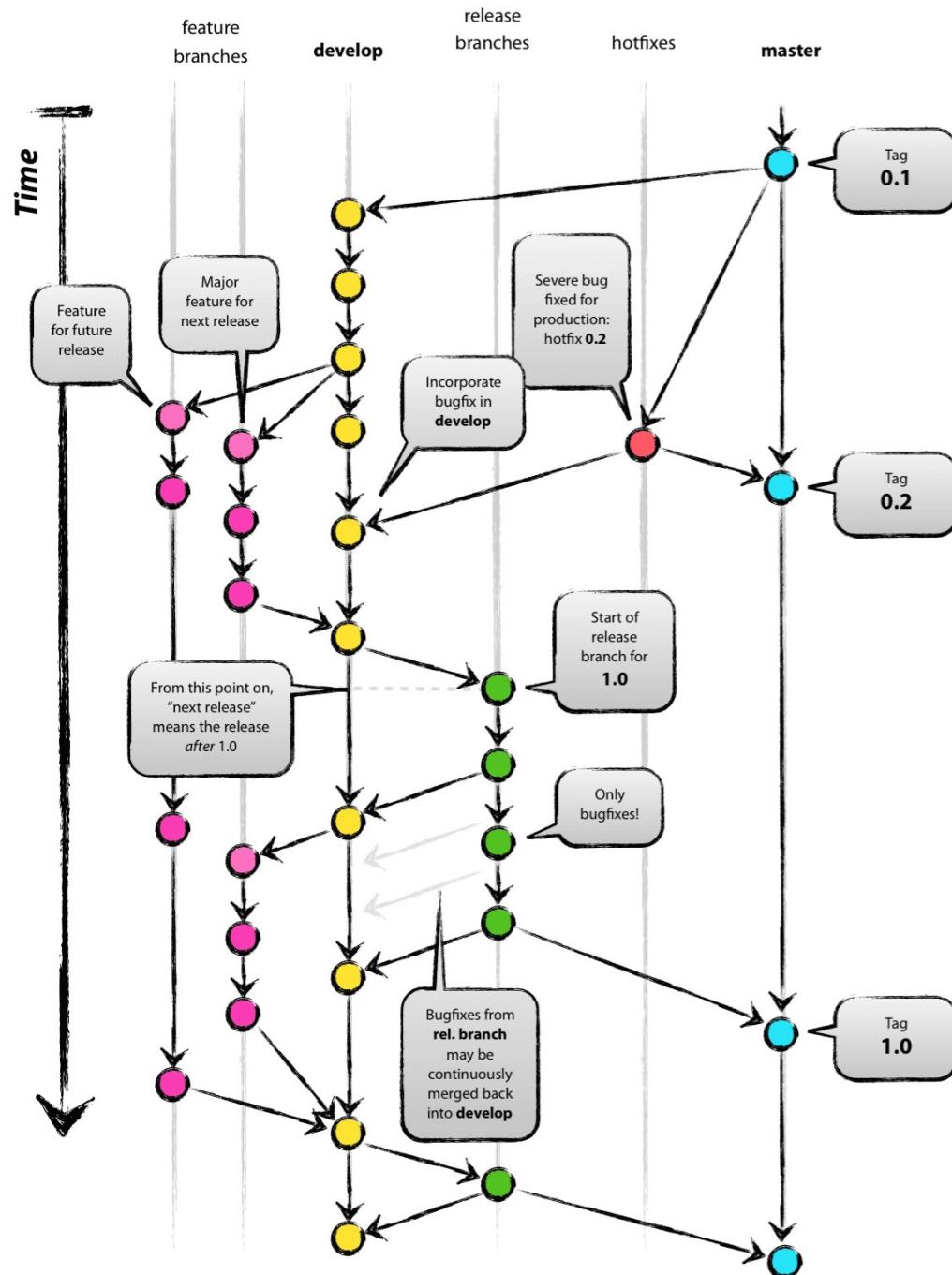
- Keine kontrollierte Vorgehensweise → schwer nachvollziehbar
- Releases entstehen an allen möglichen und unmöglichen Stellen
- Nicht so selten wie man meinen sollte (entnommen Jez Humble, „Continuous Delivery“ als Bsp. eines realen Projektes)

Branching- & Release-Strategie



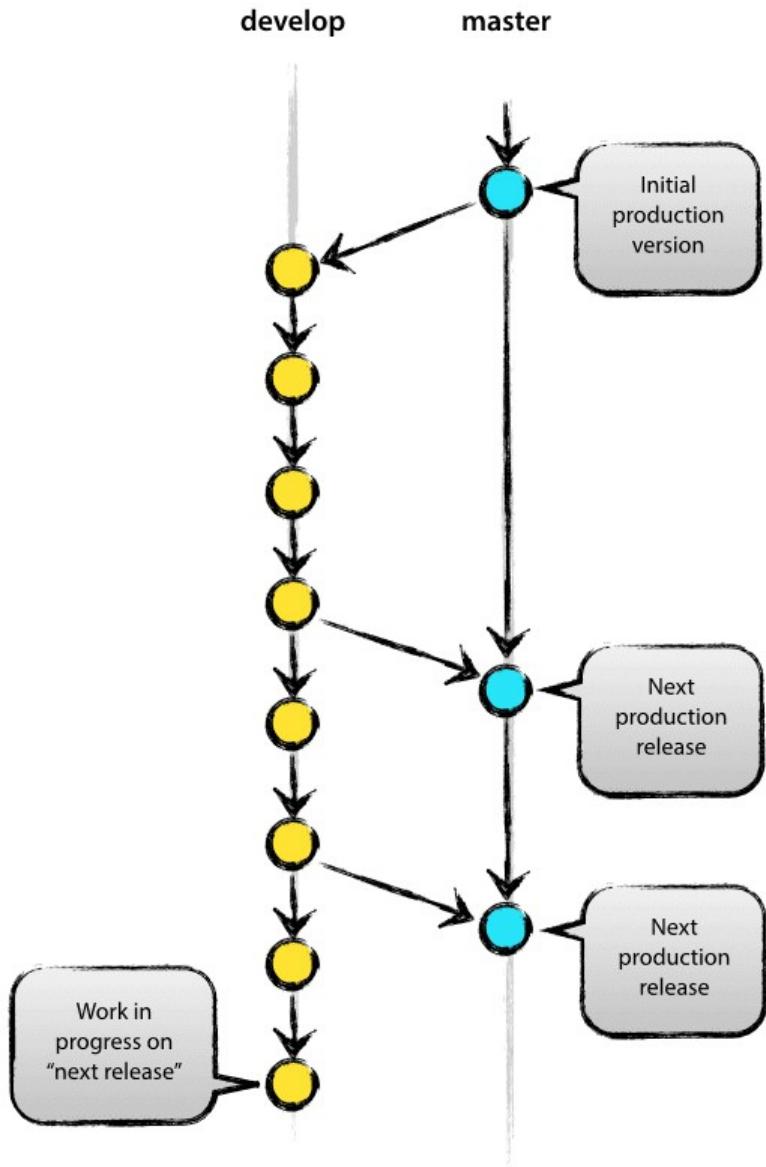
- Einzelne Release-Banches zweigen aus dem **master** ab und liefern eine klar definierte, nachvollziehbare „Release-Geschichte“
- Release-Zweige sind mehr oder weniger langlebig – je nach Supportzeitraum der einzelnen Release-Versionen
- **master** selbst ist auch immer „Deployable“ und wird nicht „gebrochen“

... bis hin zu GitFlow



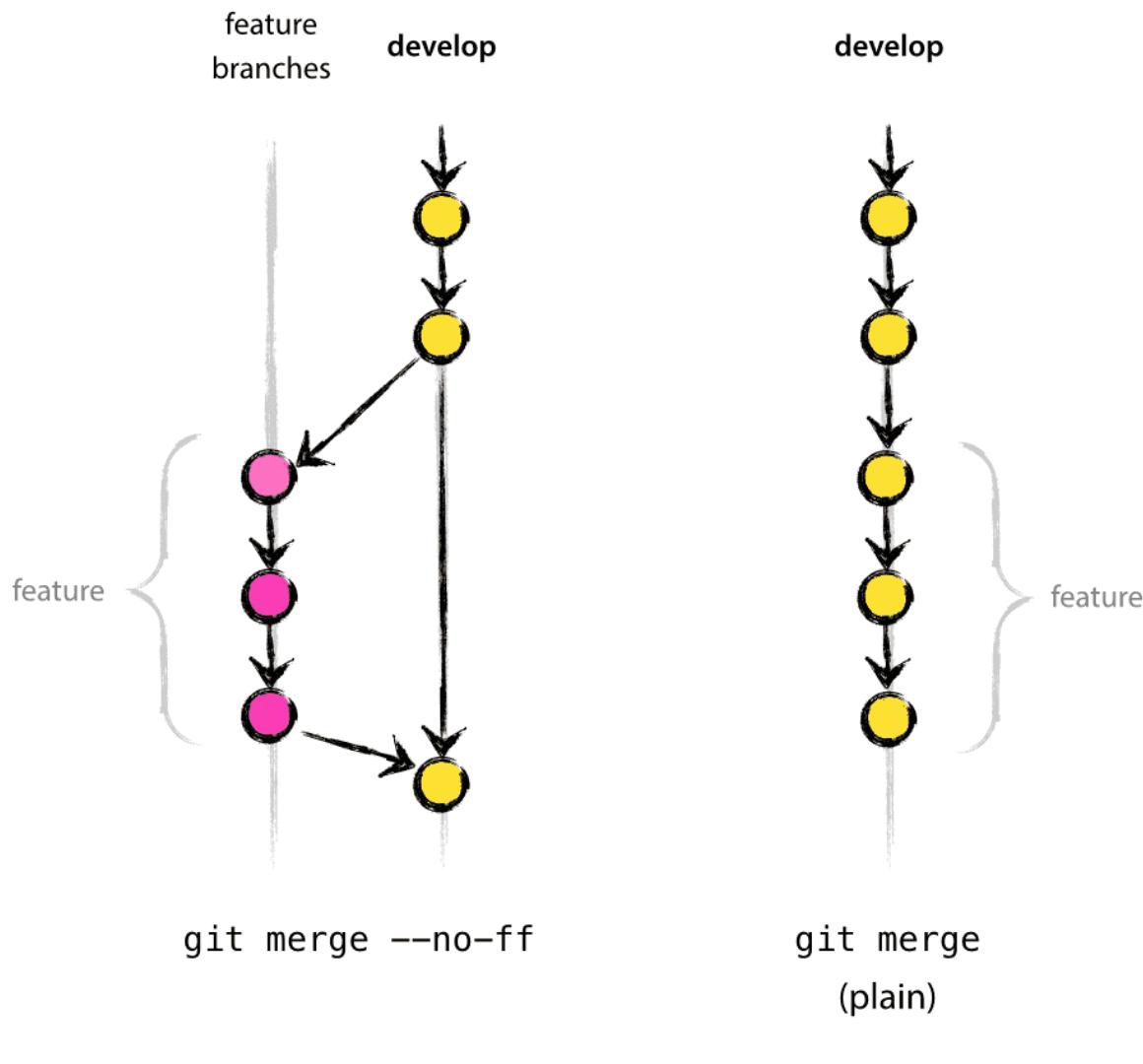
- Relativ komplexer Workflow im Vergleich zu den bisher vorgestellten Varianten
- 2010 vorgeschlagen von Vincent Driessen
- Relativ breite Unterstützung, z.B. in den GUI-Clients **SourceTree** von Atlassian (BitBucket) und **SmartGit** von syntevo
- Aber nicht unumstritten, siehe z.B. „GitFlow considered harmful“, <http://endoflineblog.com/gitflow-considered-harmful>
 - Zu schwergewichtig → macht Historie schwer nachvollziehbar
 - Unnötig komplex ... etc.
 - Abwandlungen, wie GitHub-Flow und GitLab-Flow

... bis hin zu GitFlow



- **master** und **develop** als die beiden langlebigen Hauptzweige
- Auf **master** befinden sich nur Release-Commits, d.h. jedes Release kann mit genau einem Commit auf **master** identifiziert werden
- Auf **develop** findet die ständige Entwicklung statt

... bis hin zu GitFlow



- 3 unterstützende Branches:
 - Feature-Branch
 - Release-Branch
 - Hotfix-Branch
- Feature-Branches:
 - Starten in **develop** und werden dorthin zurückgemergt
 - Günstig ist das Verwenden eines geeigneten Präfixes, wie z.B. „**feature**-“ oder „**feature**/“

... bis hin zu GitFlow



- Hotfix-Branche:
 - starten im **master** und werden sowohl in **master** wie auch in **develop** gemitert
 - Konvention ist ein entsprechender Präfix wie „**hotfix-**“ oder „**hotfix/**“
- Release-Branche:
 - starten in **develop** und werden sowohl in den **master** wie auch den **develop** gemitert
 - Konvention ist ein entsprechender Präfix wie „**release-**“ oder „**release/**“

3.5 Git anpassen

- Git lässt sich global über *git config* konfigurieren wie z.B.:

git config --global user.name „Max Mustermann“

git config --global user.email maxmu@bsp.com

- Git hat 3 Konfigurationslevel:

Level	Option	File
System	--system	/etc/gitconfig
Global	--global	~/.gitconfig
Lokal (default)	--local	.git/config

Git konfigurieren

- Gesetzte Werte auf einer niedrigeren Ebene überschreiben die auf höherer Ebene
- Konfigurationsfiles sind reiner Text → können auch im Editor verändert werden

```
1 [core]
2   repositoryformatversion = 0
3   filemode = false
4   bare = false
5   logallrefupdates = true
6   symlinks = false
7   ignorecase = true
8   hideDotFiles = dotGitOnly
9 [branch "develop"]
10    remote = github
11    merge = refs/heads/develop
12 [branch "master"]
13    remote = github
14    merge = refs/heads/master
15 [remote "local"]
16    url = C:\\\\Users\\\\Janko\\\\Documents\\\\Projekte\\\\GitStuff\\\\TempTest.git
17    fetch = +refs/heads/*:refs/remotes/local/*
```

Ausschließen von Files

- Arten von bestimmten Files oder ganze Pfade lassen sich durch Patterns in der `.gitignore` Datei ausschließen:
 - Global:
`config --global core.excludesfile ~/.gitignore_global`
 - Per Repository über lokale `.gitignore` Datei
- Guter Ausgangspunkt, wenn man ein entsprechendes File für sein Projekt sucht:<https://github.com/github/gitignore>
oder Zusammenstellen lassen via
<https://www.toptal.com/developers/gitignore>

Beispiel Patterns

- ***.exe** → Ignorieren aller ausführbaren Dateien
- **!nuget.exe** → augenommen NuGet.exe
- **/package** → Datei package im aktuellen Verzeichnis ignorieren
- **build/** → alle Dateien im Verzeichnis *build* ignorieren
- **misc/*.txt** → alle Text-Dateien unterhalb im Verzeichnis misc ignorieren, aber nicht in anderen Verzeichnissen
- **misc/**/*.pdf** → ignoriert alle PDFs im ganzen Verzeichnisbaum *misc*
-

```
1 ## Ignore Visual Studio temporary files, build results, and
2 ## files generated by popular Visual Studio add-ons.
3 ##
4 ## Get latest from https://github.com/github/gitignore/blob/master/VisualStudio.gitignore
5
6 # User-specific files
7 *.suo
8 *.user
9 *.userosscache
10 *.sln.docstates
11
12 # User-specific files (MonoDevelop/Xamarin Studio)
13 *.userprefs
14
15 # Build results
16 [Dd]ebug/
17 [Dd]ebugPublic/
18 [Rr]elease/
19 [Rr]eleases/
20 x64/
21 x86/
22 bld/
23 [Bb]in/
24 [Oo]bj/
25 [Ll]og/
26
27 # Visual Studio 2015/2017 cache/options directory
28 .vs/
29 # Uncomment if you have tasks that create the project's static files in wwwroot
30 #wwwroot/
31
32 # Visual Studio 2017 auto generated files
33 Generated\ Files/
34
35 # MSTest test Results
36 [Tt]est[Rr]esult*/
37 [Bb]uild[Ll]og.*
```

Beispiel *.gitignore* für Visual Studio Projekte

.gitignore gilt von dem Verzeichnis in dem es ist an „abwärts“ → verschachtelte *.gitignore* Strukturen sind möglich

Git Attribute

- Abgelegt in
 - *.gitattributes* Datei in einem Verzeichnis (meistens im Wurzelverzeichnis des Repositorys → Attribute-File kommt mit ins Repository)
 - *.git/info/attributes* → Attribut-File wird nicht ins Repository committed
- Spezifikation von
 - Verschiedene Merge-Strategien für unterschiedliche File-Typen
 - Spezifische Diff-Methoden für nicht-textbasierte Datenfiles angeben
 - Inhaltsbasierte Filter einstellen

Git Attribute – Binärfils

- Manche Dateien sehen zwar wie Text-Files aus sollten aber wie binäre behandelt werden, sollen z.B. Files mit der Extension CMA als binär behandelt werden:
**.cma binary*
- Spezieller (EXIF-)Diff für binäre Dateien, z.B. PNG-Bilder:
**.png diff=exif*
 - „*exif*“-Filter soll für *.png Dateien verwendet werden
 - *git config diff.exif.textconv exiftool*
 - git diff zeigt nun die Änderung der EXIF-Daten an

Git Attribute – Spezifische Vergleiche (sematische Diff's)

The screenshot shows the GitHub page for the Difftastic project. The page features a large "it's difftastic!" heading with "difftastic" in green. Below it are links for manual (en), manual (zh-CN), crates.io (v0.52.0), and coverage (82%). A note says "Difftastic is a structural diff tool that compares files based on their syntax." and provides installation instructions. A "Basic Example" section shows a terminal window running the command \$ difft javascript_simple_before.js javascript_simple_after.js. The terminal output shows a semantic diff between two JavaScript files, highlighting changes in indentation, line alignment, and line wrapping.

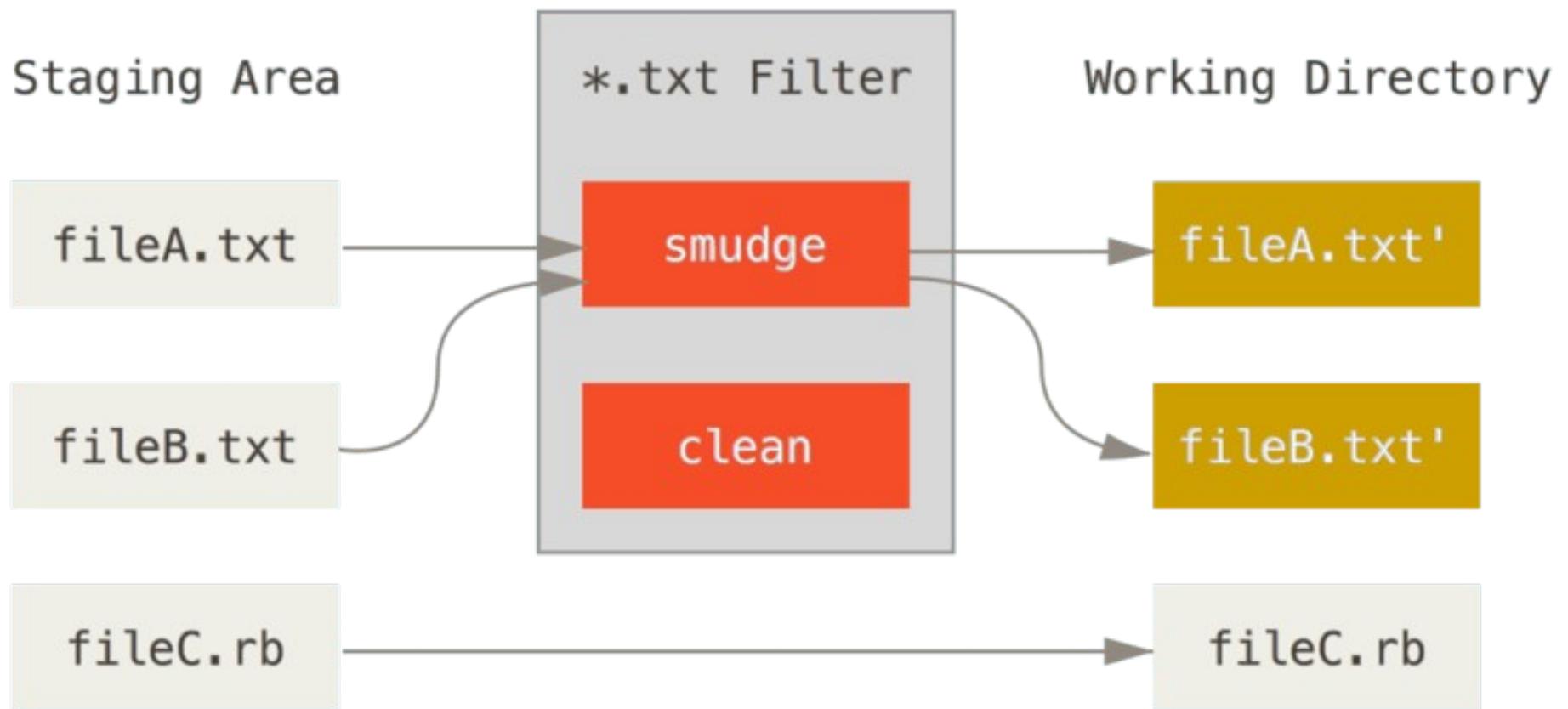
```
$ difft javascript_simple_before.js javascript_simple_after.js
javascript_simple_after.js --- JavaScript
1 // hello
.
2 foo();
3 bar(1);
4 baz();
.
5
6 var people = [
7   "john", "harry", "dick", "eric",
8   "jenny", "alexandra",
9 ];
10
11 ];

$
```

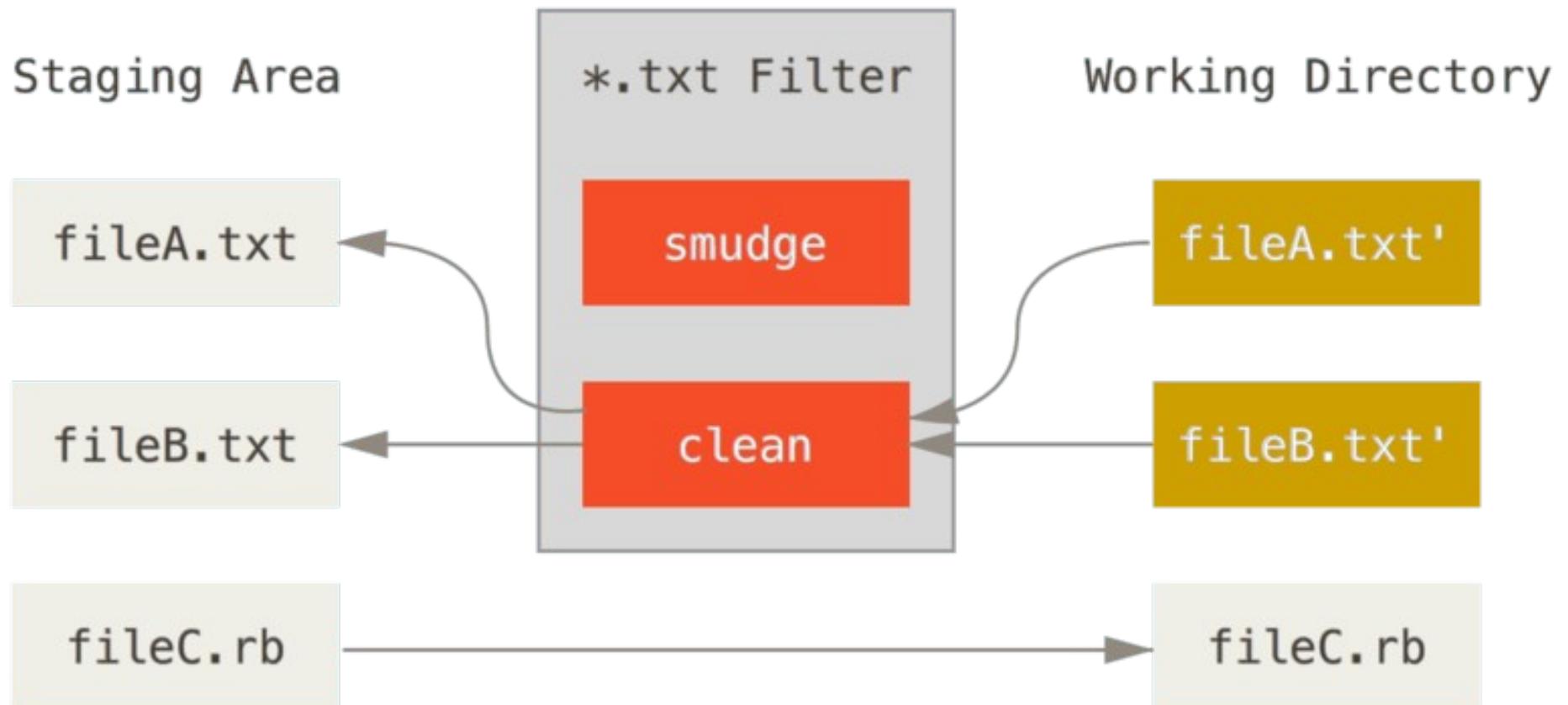
In this JavaScript example, we can see:

- (1) Difftastic understands nesting. It highlights the matching `{` and `}`, but understands that `foo()` hasn't changed despite the leading whitespace.
- (2) Difftastic understands which lines should be aligned. It's aligned `bar(1)` on the left with `bar(2)` on the right, even though the textual content isn't identical.
- (3) Difftastic understands that line-wrapping isn't meaningful. `"eric"` is now on a new line, but it hasn't changed.

Git Checkout – *Smudge-Filter*



Git Checkout – *Clean-Filter*



Smudge- und Clean-Filter – Beispiel Jupyter-Notebook

- Jupyter-Notebook-Format → JSON-Format, das die Ergebnisse einzelner Code-Abschnitte enthält → ungünstig für die Source-Code Verwaltung → Entfernen der Ergebnisteile
- Filter:
**.ipynb filter=nbstrip_full*
- Eintrag des Filters in *.git/config* im *.git*-Verzeichnis:
[filter "nbstrip_full"]

```
clean = "C:/Users/username/anaconda3/envs/ProgDatSci/python  
C:/Users/username/anaconda3/envs/ProgDatSci/Lib/site-packages/nbstri  
pout"
```

smudge = cat

required = true

Git Hooks – Skripte

- Bestimmte Punkte im Git-Ablauf lassen sich dadurch mit zusätzlicher Funktionalität per Hook-Skript konfigurieren
- Liegen im /hooks Ordner des Repository-Verzeichnisses
 - Client-Side Hooks → lokales Repository
 - Server-Side Hooks → Bare-Repository
- Namen der Skripte sind vordefiniert und mit bestimmten Punkten im Git-Ablauf verbunden

Git Hooks – Skripte

- Nach der Initialisierung des Git-Repositorys liegen Beispiel-Skripte im *hooks* Verzeichnis

```
dietzs@JANKO-PC MINGW64 ~/Documents/temp/test/.git/hooks (GIT_DIR!)
```

```
$ ls -al
total 44
drwxr-xr-x 1 dietzs 1049089      0 Apr 19 12:36 .
drwxr-xr-x 1 dietzs 1049089      0 Apr 19 12:36 ../
-rw xr-xr-x 1 dietzs 1049089 478 Apr 19 12:36 applypatch-msg.sample*
-rw xr-xr-x 1 dietzs 1049089 896 Apr 19 12:36 commit-msg.sample*
-rw xr-xr-x 1 dietzs 1049089 3327 Apr 19 12:36 fsmonitor-watchman.sample*
-rw xr-xr-x 1 dietzs 1049089 189 Apr 19 12:36 post-update.sample*
-rw xr-xr-x 1 dietzs 1049089 424 Apr 19 12:36 pre-applypatch.sample*
-rw xr-xr-x 1 dietzs 1049089 1642 Apr 19 12:36 pre-commit.sample*
-rw xr-xr-x 1 dietzs 1049089 1492 Apr 19 12:36 prepare-commit-msg.sample*
-rw xr-xr-x 1 dietzs 1049089 1348 Apr 19 12:36 pre-push.sample*
-rw xr-xr-x 1 dietzs 1049089 4898 Apr 19 12:36 pre-rebase.sample*
-rw xr-xr-x 1 dietzs 1049089 544 Apr 19 12:36 pre-receive.sample*
-rw xr-xr-x 1 dietzs 1049089 3610 Apr 19 12:36 update.sample*
```

Bsp. Server-side *post-receive* Hook Skript

```
#!/bin/bash -x

JENKINSURL="http://build.corporation.com:8080

JOB="MyProject"
TOKEN="JenkinsToken"

# read all commit ids and refnames from stdin
# but only keep the last one
while read oldrev newrev refname
do
    REV=$newrev
    BRANCH=$(git rev-parse --symbolic --abbrev-ref $refname)
done
REV=${REV:0:7}

COMMITTER_EMAIL=$(git log -1 --format=format:%aE $REV)

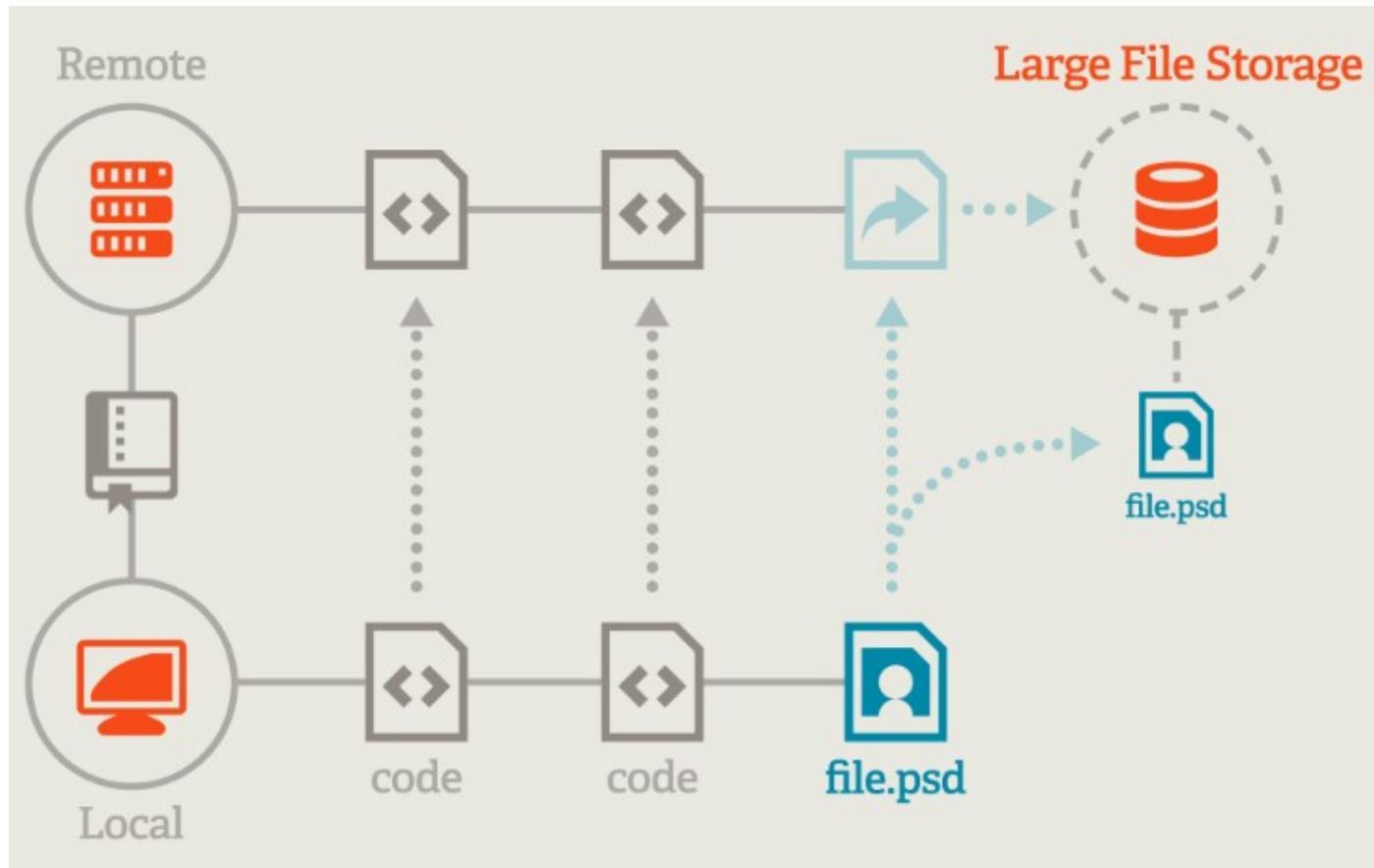
if [ "$BRANCH" == "master" -o "$BRANCH" == "develop" ]; then
    # trigger the initial pipeline job

    URL="${JENKINSURL}/job/${JOB}/buildWithParameters?token=${TOKEN}&SHA=${REV}&TYPE=${BRANCH}&COMMIT
TER_EMAIL=${COMMITTER_EMAIL}"
    #echo "URL: ---$URL---" >> log-file.txt
    /usr/bin/curl -u jenkins-user:key-for-jenkins-user "${URL}"
fi
```



Git LFS (Large File System)

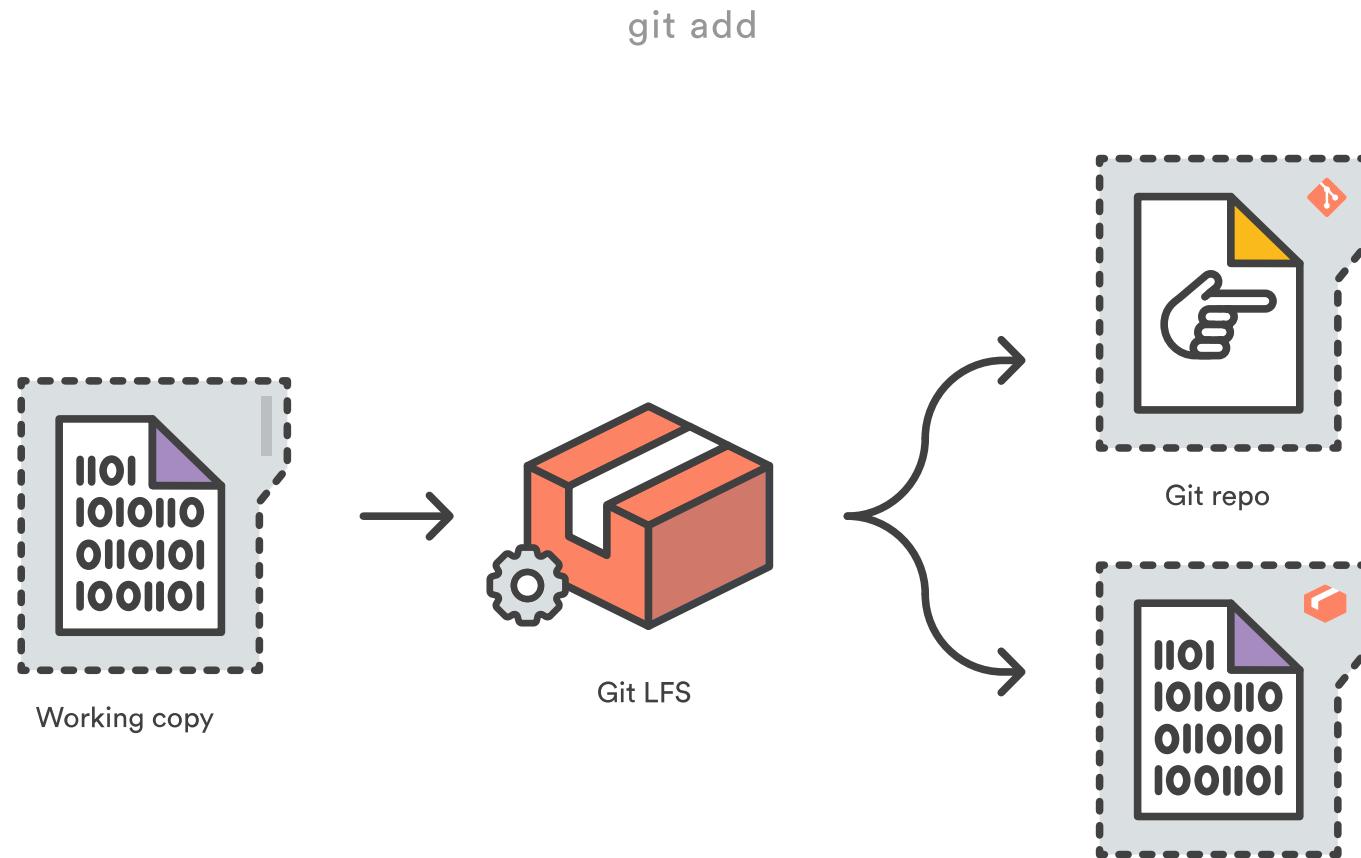
- Lösung für das Problem großer Binärfiles im Repository:





Git LFS (*Large File System*)

- Files werden durch Pointer ersetzt, die aber transparent von Git LFS behandelt werden

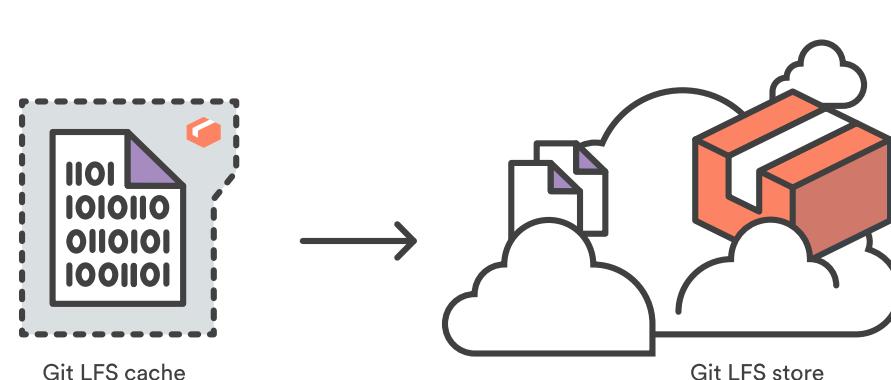
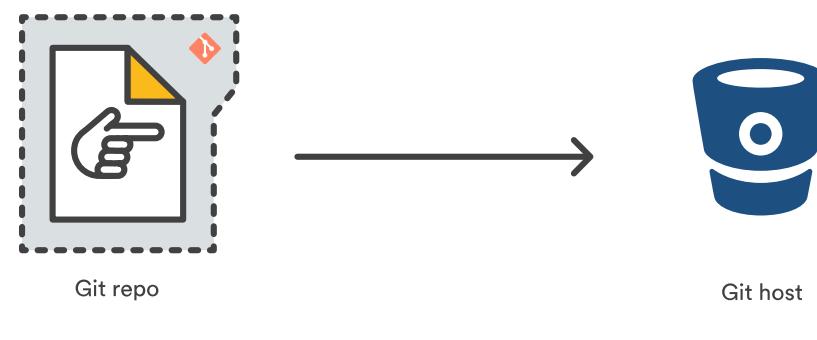




Git LFS (Large File System)

- Wird gepusht, wird der Pointer auf das binäre File im Repository abgelegt und das File selbst im LFS-Store

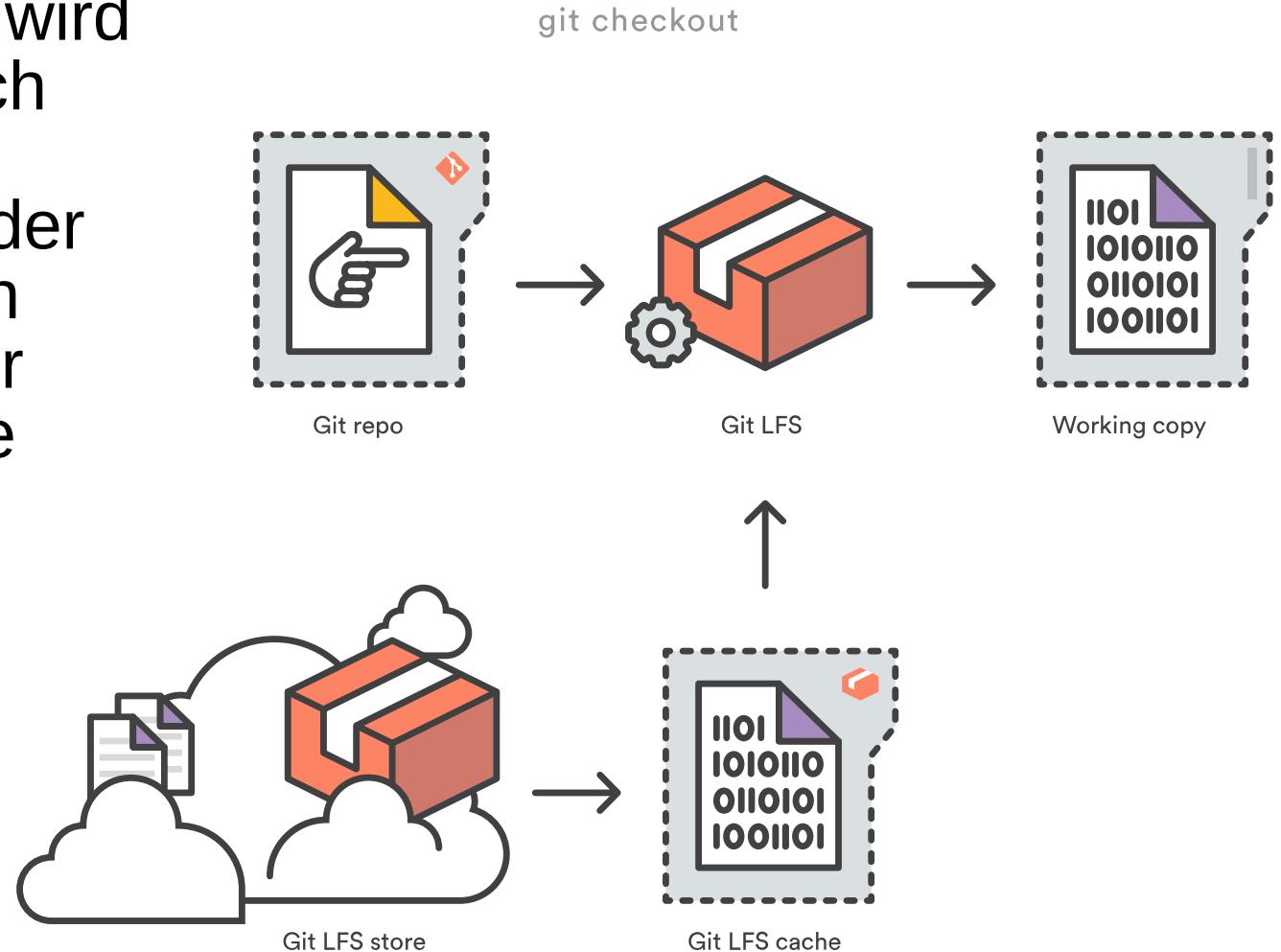
git push





Git LFS (Large File System)

Beim Checkout wird der Pointer durch das binäre File ersetzt – entweder aus dem lokalen LFS-Cache oder aus dem remote LFS-Store runtergeladen





Git LFS (*Large File System*)

- Muß (nur) einmal im System installiert werden:
git lfs install
- Git LFS aber pro Repository initialisieren:
git init
git lfs install
 Installiert im Repository einen speziellen *pre-push* Hook
- Angeben welche Files per LFS zu tracken sind:
*git lfs track *.ogg*
 Patterns analog zu *.gitignore*, aber keine „negative Patterns“ möglich



Git LFS (*Large File System*)

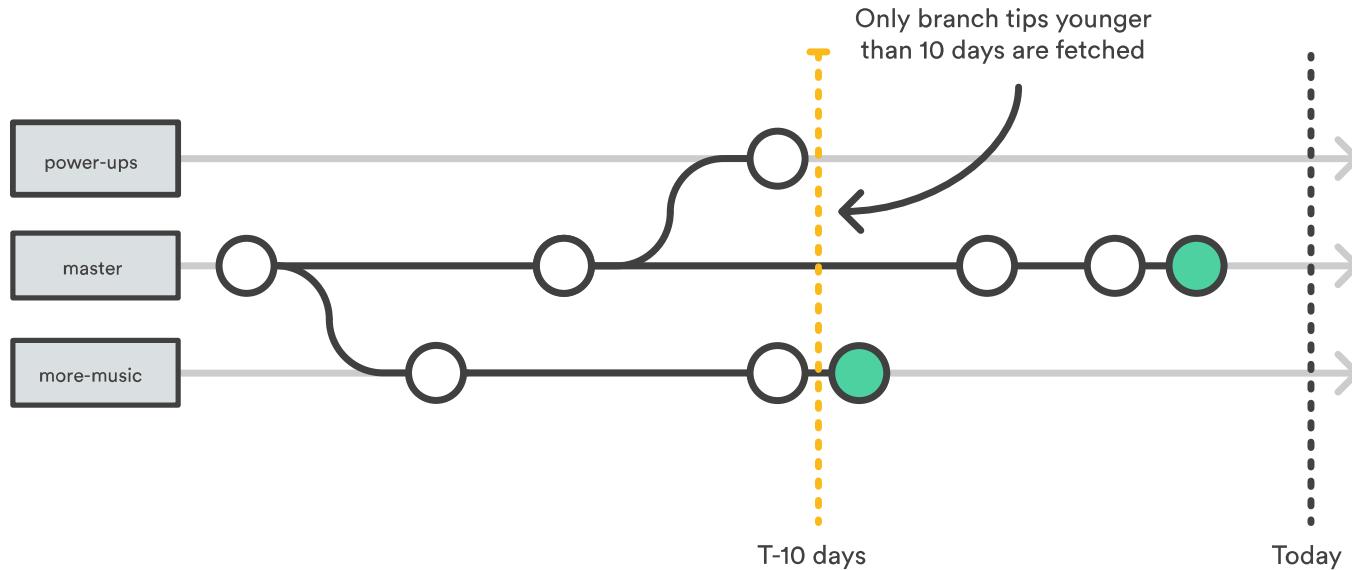
- Setzt entsprechenden LFS-Filter im `.attributes` File:
`*.ogg filter=lfs diff=lfs merge=lfs -text`
- Rücksetzen des Trackings über `untrack`
- Commiten und Pushen wie gewohnt
- Lädt Datei für die aktuell ausgecheckte Version runter
- Herunterladbar Git-LFS History lässt sich beeinflussen
`git lfs fetch --recent`



Git LFS (Large File System)

git lfs fetch --recent

```
[lfs]  
fetch recent refs days = 10
```



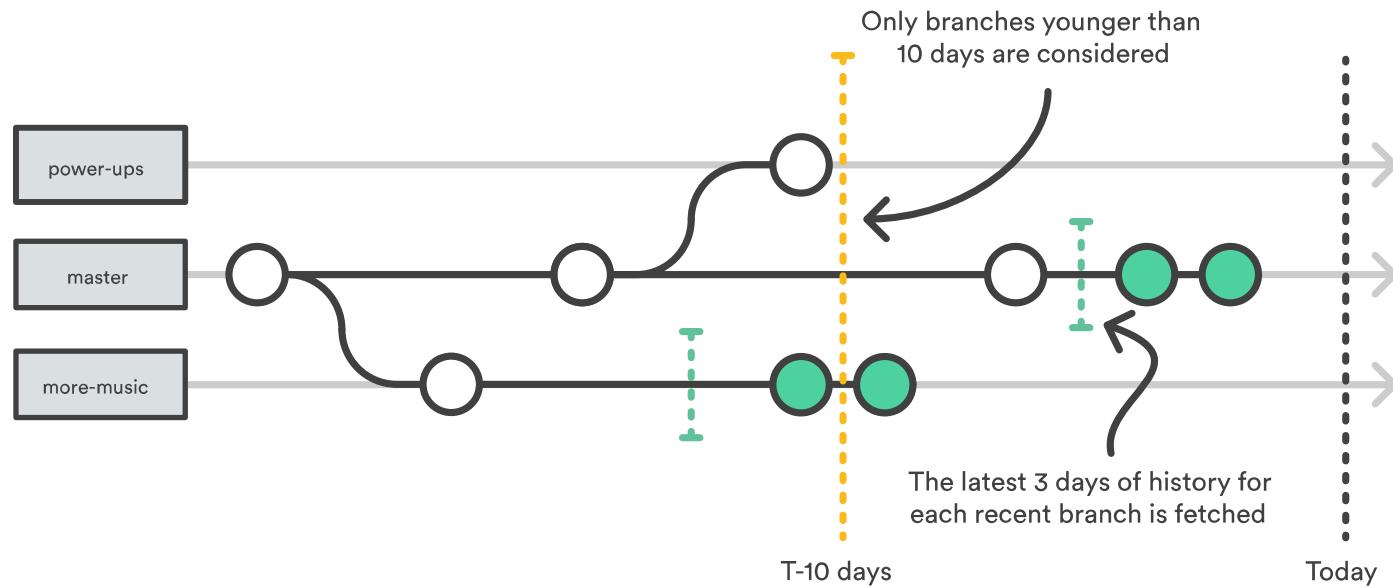
git config lfs.fetchrecentrefsdays 10



Git LFS (Large File System)

git lfs fetch --recent

```
[lfs]
fetch recent refs days = 10
fetch recent commits days = 3
```



git config lfs.fetchrecentcommitsdays 3



Git LFS (*Large File System*)

Ganze Historie holen:

git lfs fetch –all

Alte Files löschen:

git lfs prune

D.h. Files auf die kein ausgecheckter, kein noch nicht gepushter und kein „recent“ Commit zeigt

git lfs prune --verbose --dry-run

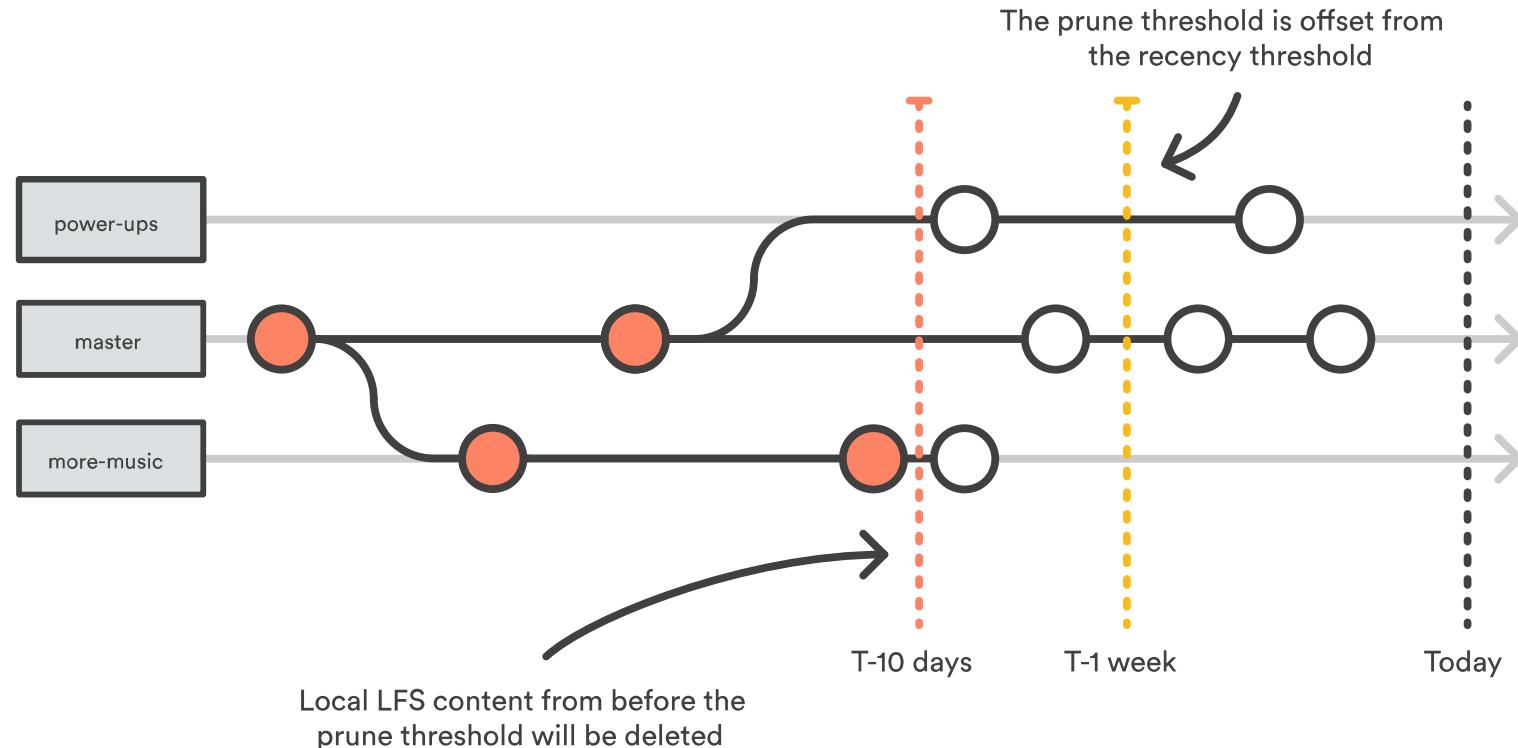


Git LFS (Large File System)

git lfs prune

[lfs]

fetch recent refs days = 7
prune offset days = 3



Git Internals

Plumbing – Porcelain

hash-object
cat-file
ls-tree
mktree
commit-tree
fsck
gc
index-pack
ls-files
mktag
pack-objects
repack
verify-pack



commit
push
pull
merge
branch
checkout
clone
bisect
tag
diff

Praktische Übungen

- Projekt-Repository anlegen, Repository clonen, Bare-Repository erstellen
- History-Rewrite: ***rebase, amend, reset, cherry-pick*** ...
- Workflows: Master/Develop-Workflow, Feature/Topic Branches, Gitflow ...
- Nützliche Referenz: [A Visual Git Reference](#)
- ... und wenn etwas schief geht: [Oh shit, git!](#)
- Open Source Spiel zum Lernen von Git: [Oh My Git !](#)
- Schön illustrierte Kurzeinführung (R-lastig): [GitHub - The Perks of Collaboration and Version Control](#)