Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2022/23

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) **Git, GitHub**
5. Wissenschaftliches Rechnen
   a) NumPy, SciPy
6. Datenverarbeitung und -visualisierung
   a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
   a) Grundlegende Ansätze (Datensätze, Auswertung)
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers

Themen

I. Versionsverwaltung

II. Git

    1. Idee, Konfiguration

    2. Lokal: Commit, Stash, Branch, Merge

    3. Remote: Push, Pull, Merge

III. GitHub

I.
Versionsverwaltung

Versionen und Verlauf

- Verschiedene Versionen z.B. eines Programms
  - Verschiedene Features/ Probleme werden (gleichzeitig) bearbeitet

`data.py`

`plot.py`

Versionen und Verlauf

- Verschiedene Versionen z.B. eines Programms
  - Verschiedene Features/ Probleme werden (gleichzeitig) bearbeitet

`data.py`

`data_fixed.py`

`plot.py`

Versionen und Verlauf

- Verschiedene Versionen z.B. eines Programms
  - Verschiedene Features/ Probleme werden (gleichzeitig) bearbeitet

`data.py`

`data_fixed.py`

`20221123_data.py`

`plot.py`

Versionen und Verlauf

- Verschiedene Versionen z.B. eines Programms
  - Verschiedene Features/ Probleme werden (gleichzeitig) bearbeitet

`data.py`

`data_fixed.py`

`20221123_data.py`

`plot.py`

`plot_v2.py`

`20221001_plot.py`

Versionen und Verlauf

- Verschiedene Versionen z.B. eines Programms
  - Verschiedene Features/ Probleme werden (gleichzeitig) bearbeitet
- Verlauf soll gespeichert werden

`data.py`

`data_fixed.py`

`20221123_data.py`

`plot.py`

`plot_v2.py`

`20221001_plot.py`

Versionen und Verlauf

- Verschiedene Versionen z.B. eines Programms
  - Verschiedene Features/ Probleme werden (gleichzeitig) bearbeitet
- Verlauf soll gespeichert werden

`data.py`

`data_fixed.py`

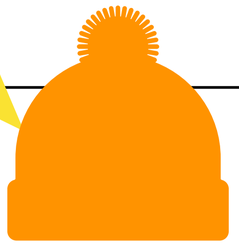
`20221123_data.py`

`plot.py`

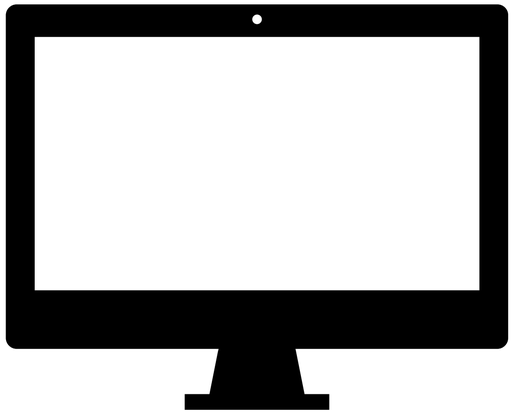
`plot_v2.py`

`20221001_plot.py`

`import` nutzt den Dateinamen :-(



Verschiedene Entwicklungsorte



Verschiedene Entwicklungsorte

Verschiedene Entwicklungsorte

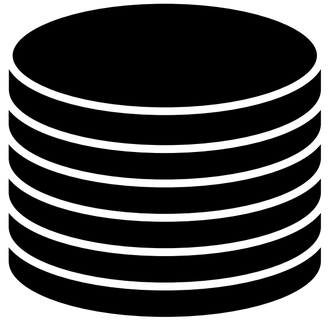
Bugfix

Feature *XI* für nächstes Release

Verschiedene Entwicklungsorte

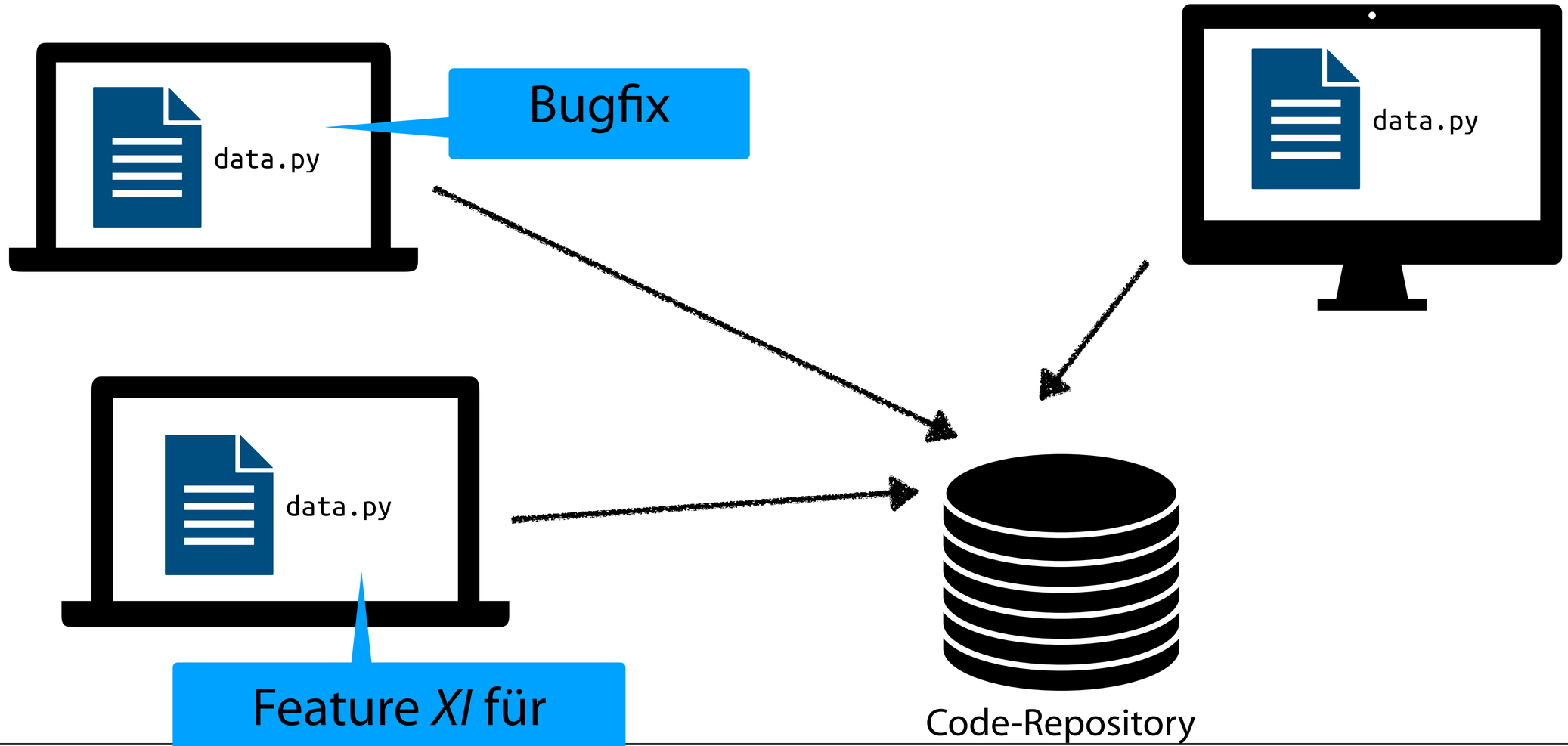
Bugfix

Feature *XI* für nächstes Release

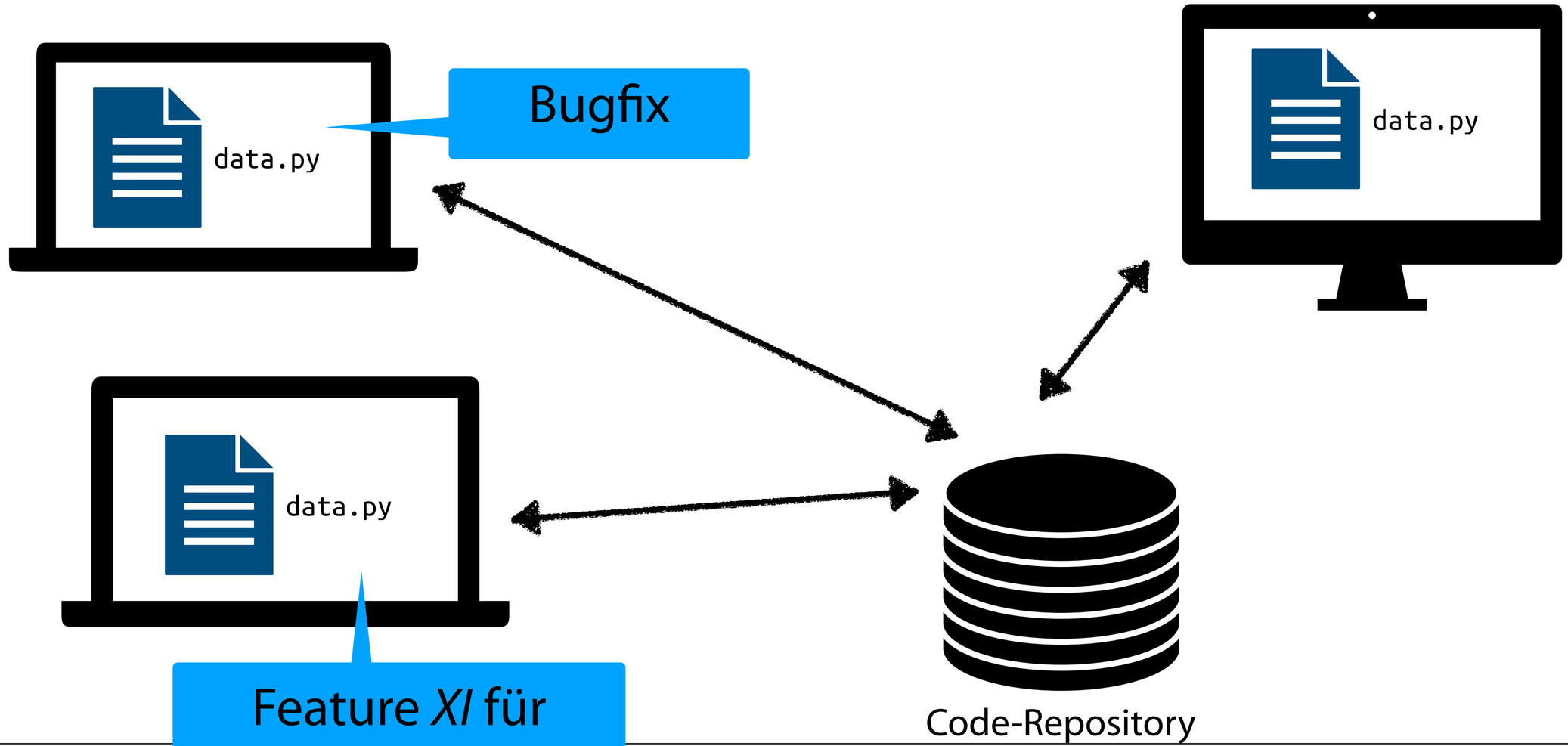


Code-Repository

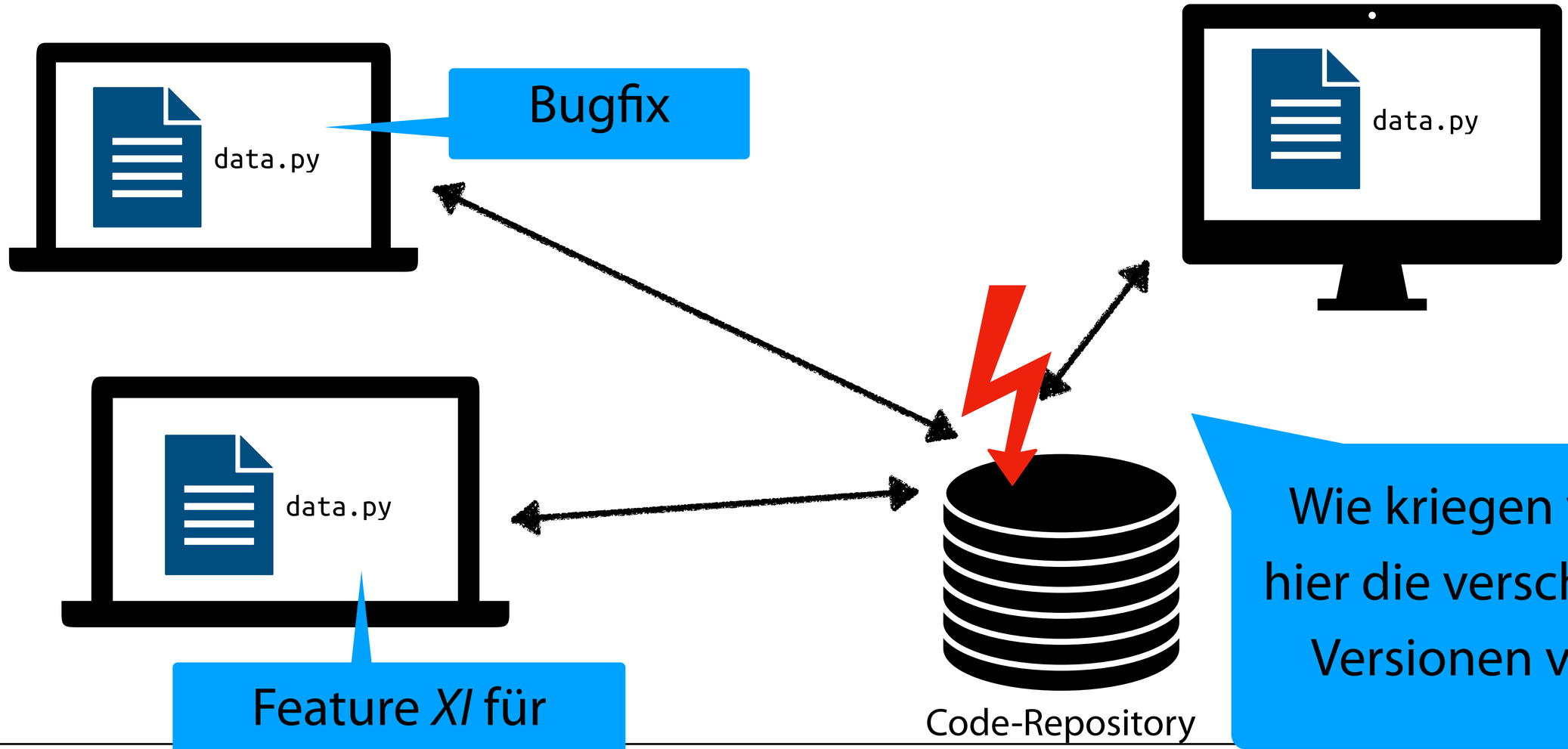
Verschiedene Entwicklungsorte



Verschiedene Entwicklungsorte



Verschiedene Entwicklungsorte

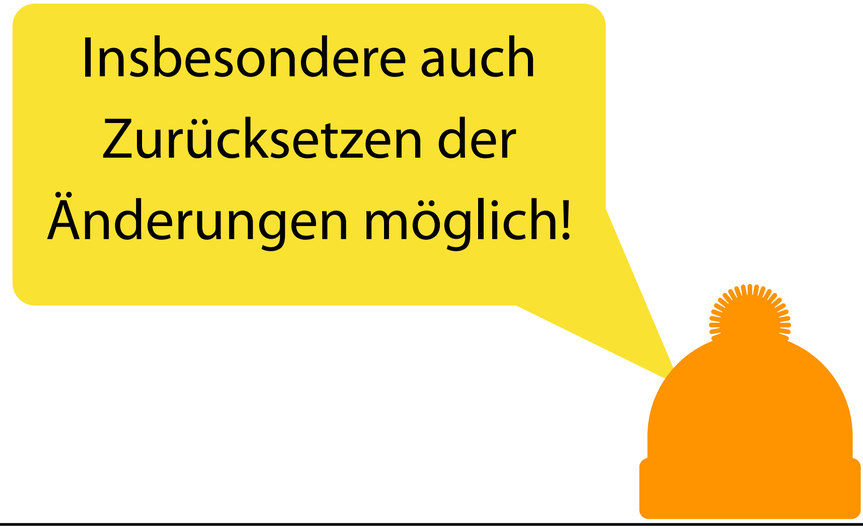


Lösung: Versionsverwaltung

- Verlauf der Änderungen (textbasierte Dateien)

Lösung: Versionsverwaltung

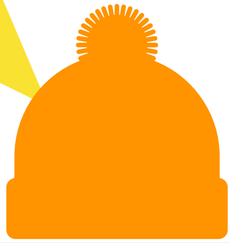
- Verlauf der Änderungen (textbasierte Dateien)



Lösung: Versionsverwaltung

- Verlauf der Änderungen (textbasierte Dateien)
- Verschiedene Entwicklungszweige gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte

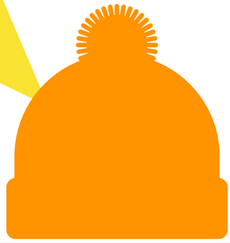
Insbesondere auch Zurücksetzen der Änderungen möglich!



Lösung: Versionsverwaltung

- Verlauf der Änderungen (textbasierte Dateien)
- Verschiedene Entwicklungszweige gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte
- Zusammenführen von Entwicklungszweigen

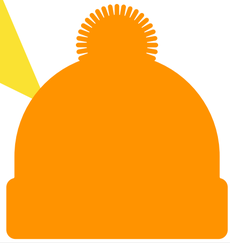
Insbesondere auch Zurücksetzen der Änderungen möglich!



Lösung: Versionsverwaltung

- Verlauf der Änderungen (textbasierte Dateien)
- Verschiedene Entwicklungszweige gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte
- Zusammenführen von Entwicklungszweigen
- Ein (zentrales) Repository

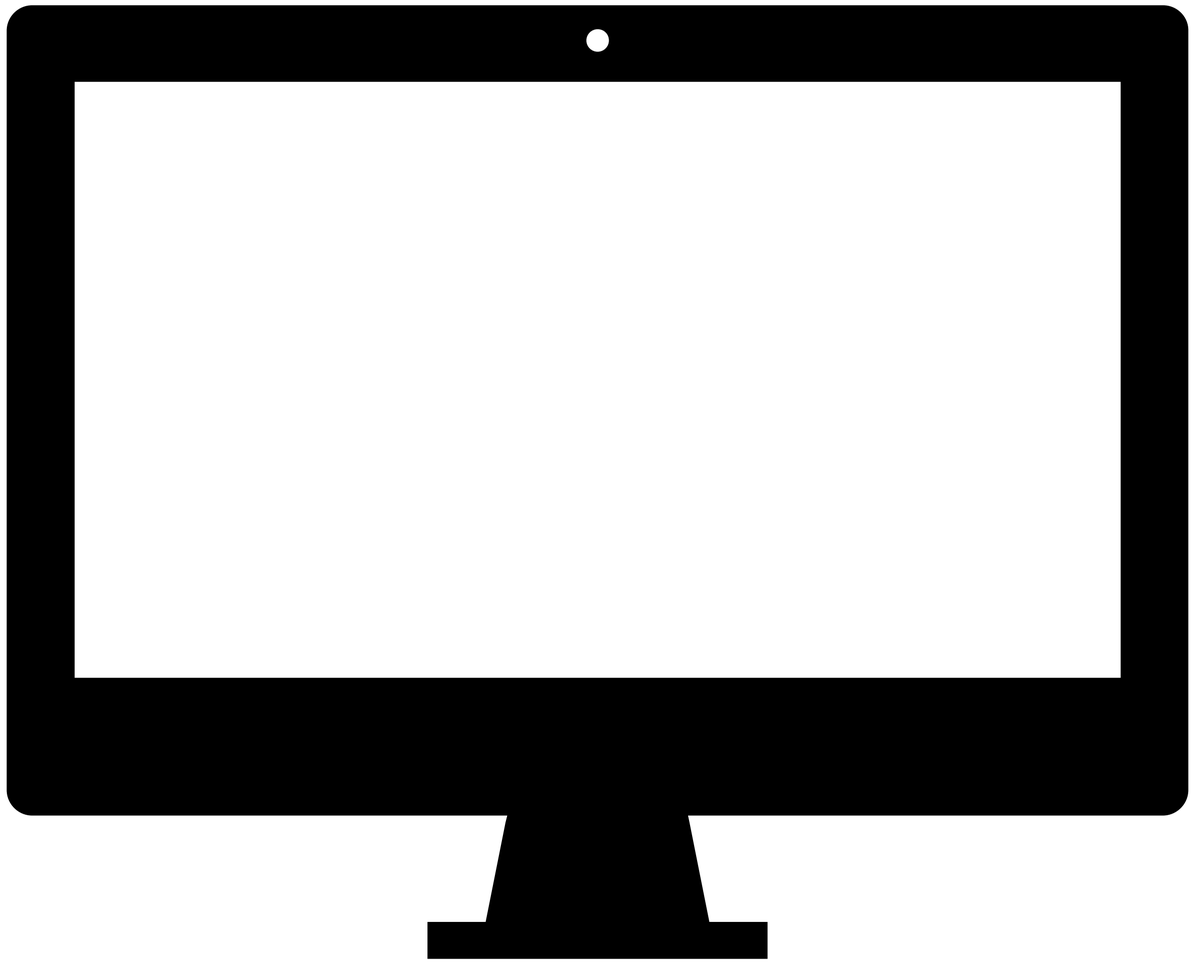
Insbesondere auch Zurücksetzen der Änderungen möglich!



II.
Git

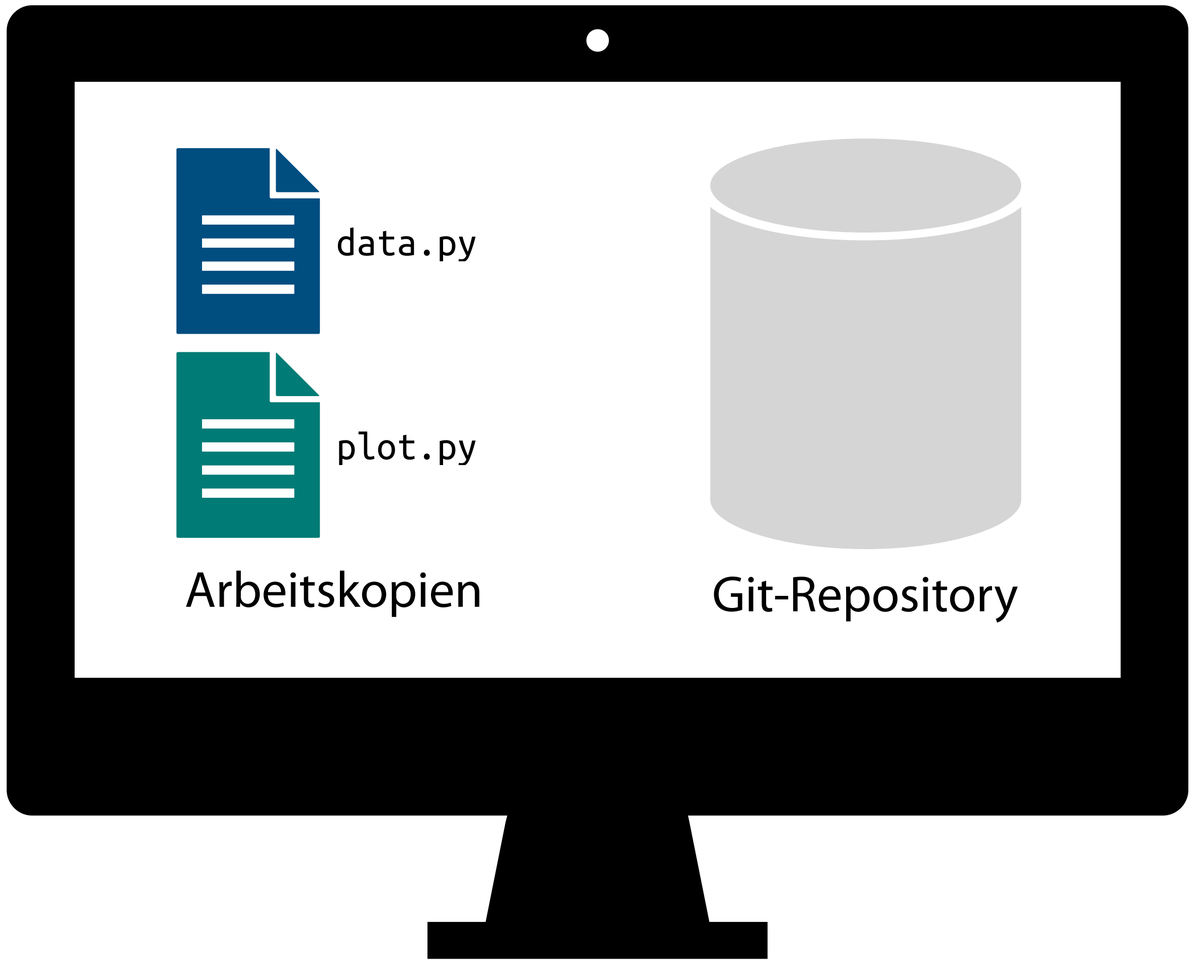
1. Idee, Konfiguration

Verteilte Repositories

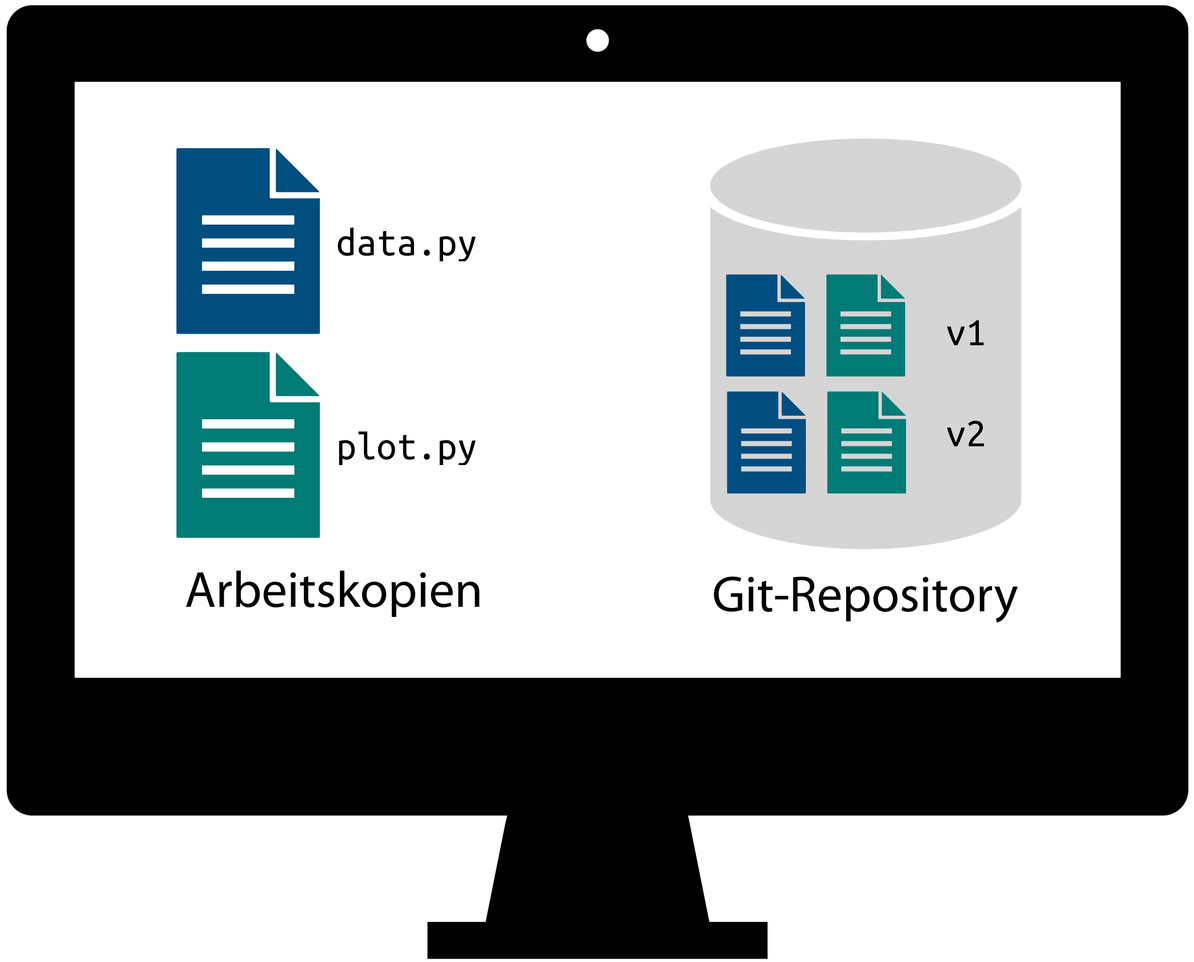


Verteilte Repositories

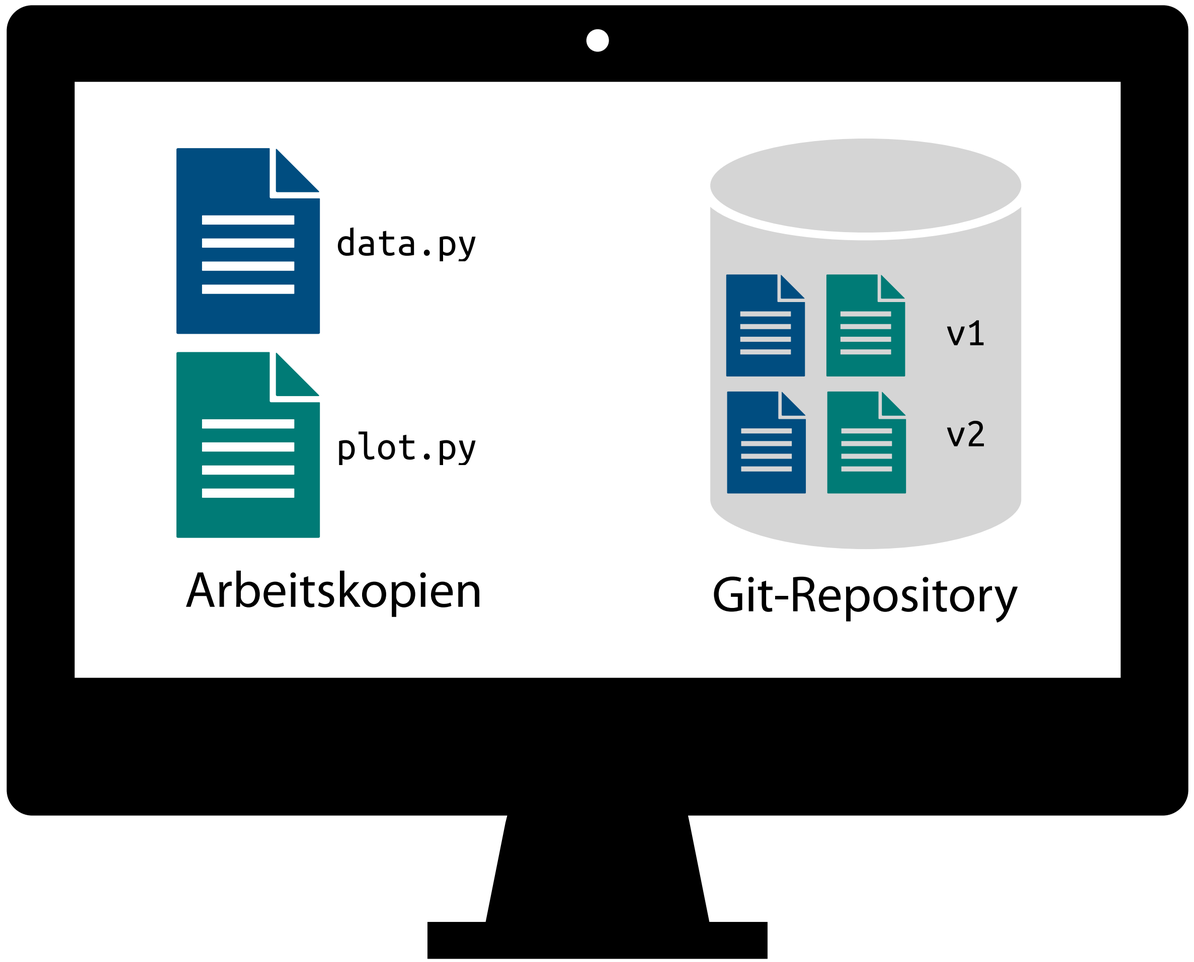
Verteilte Repositories



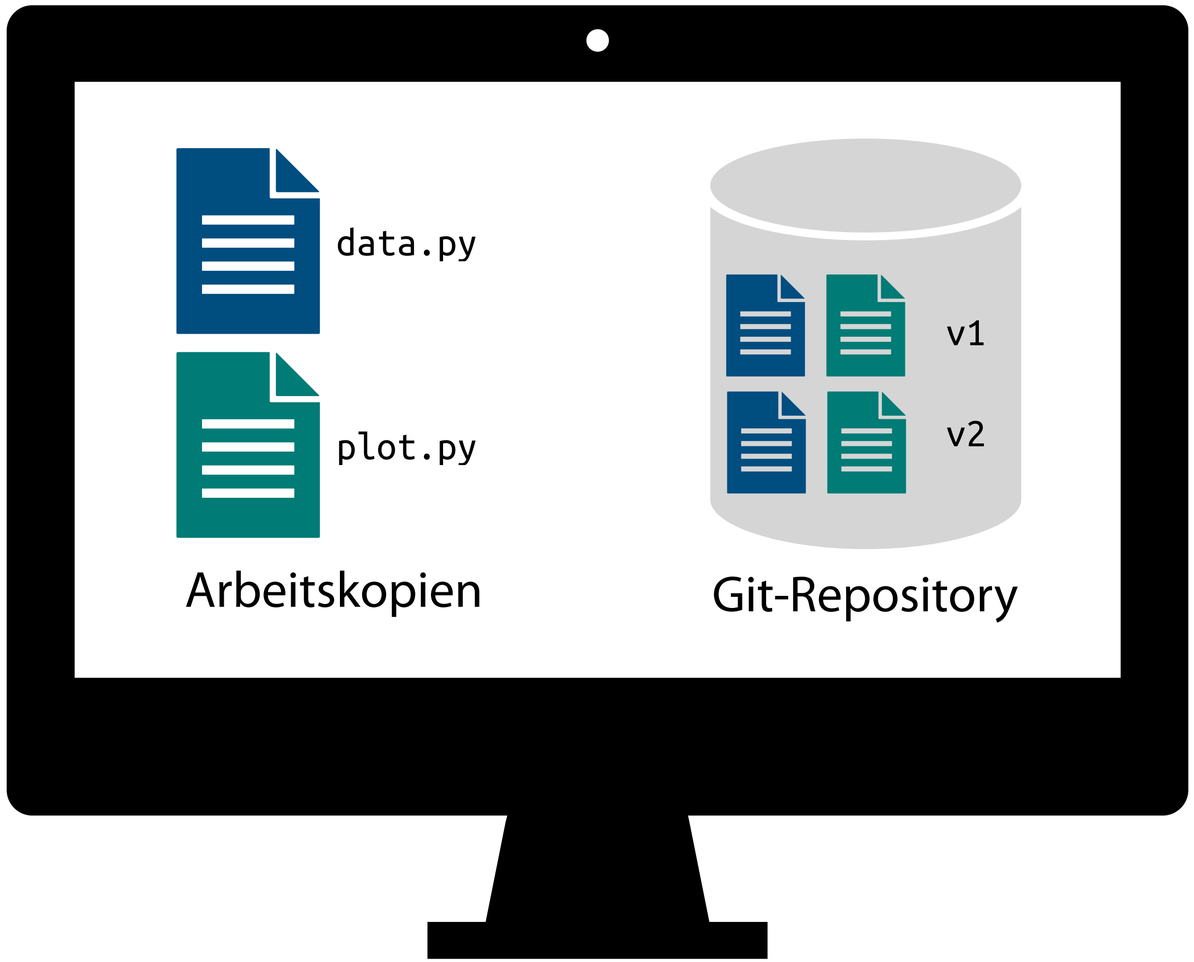
Verteilte Repositories

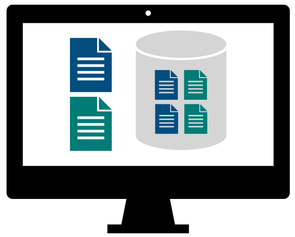


Verteilte Repositories

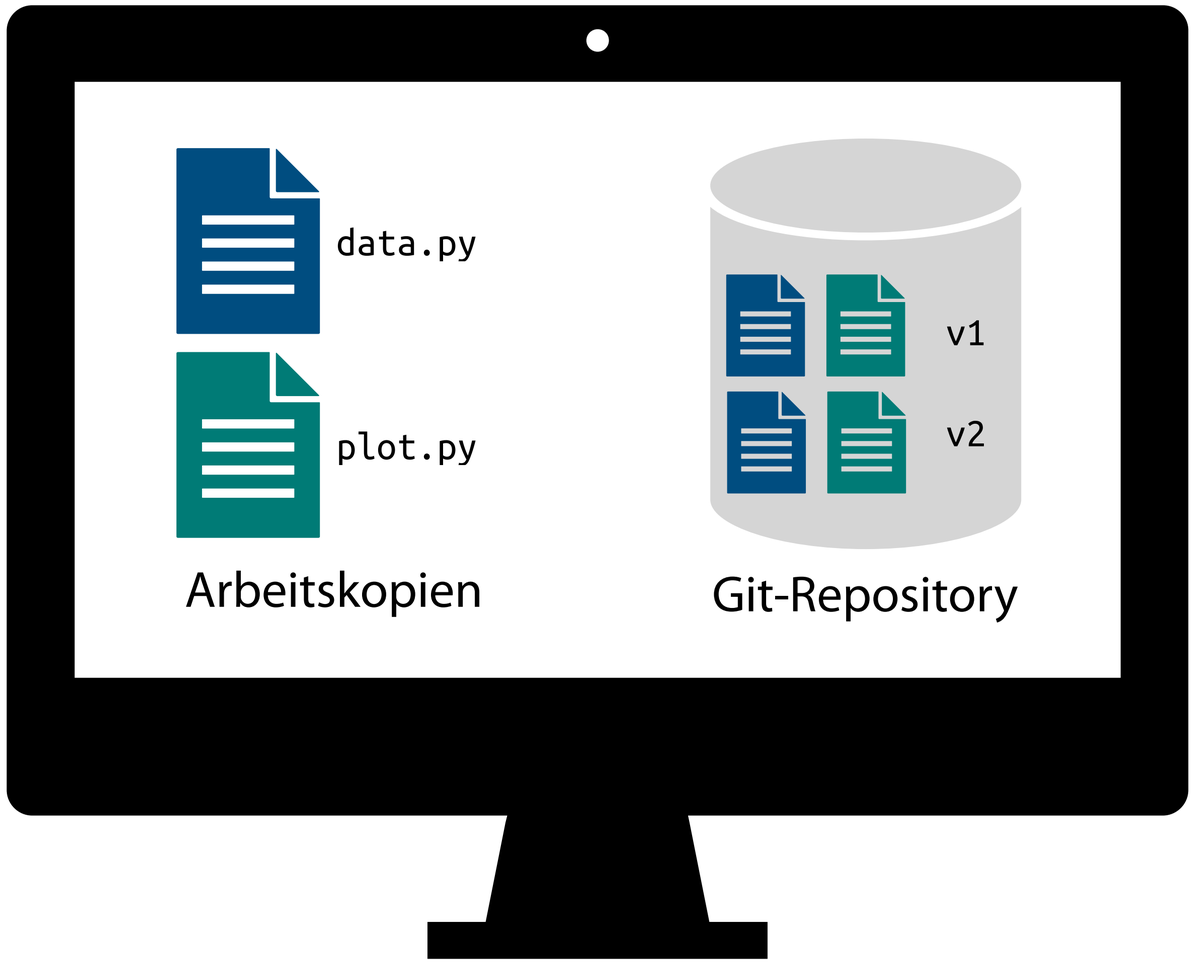


Verteilte Repositories

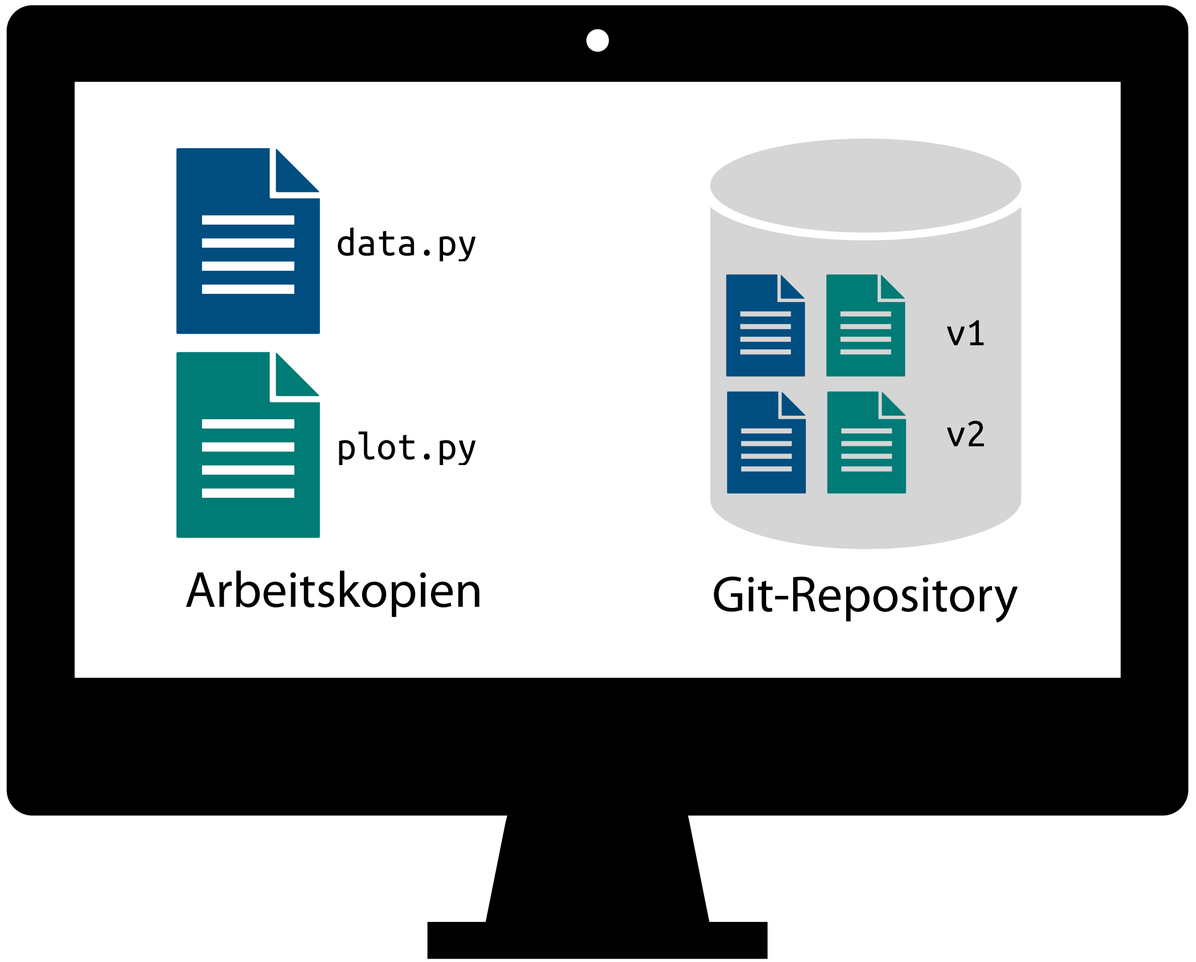


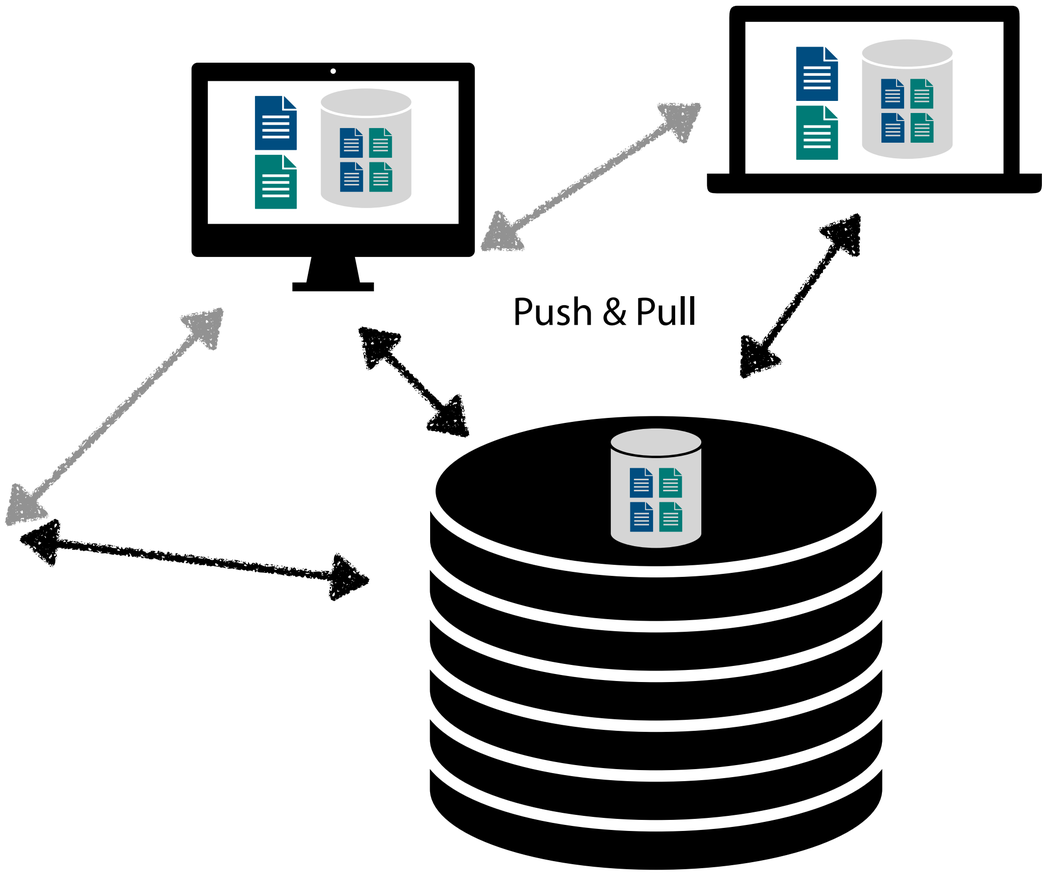


Verteilte Repositories



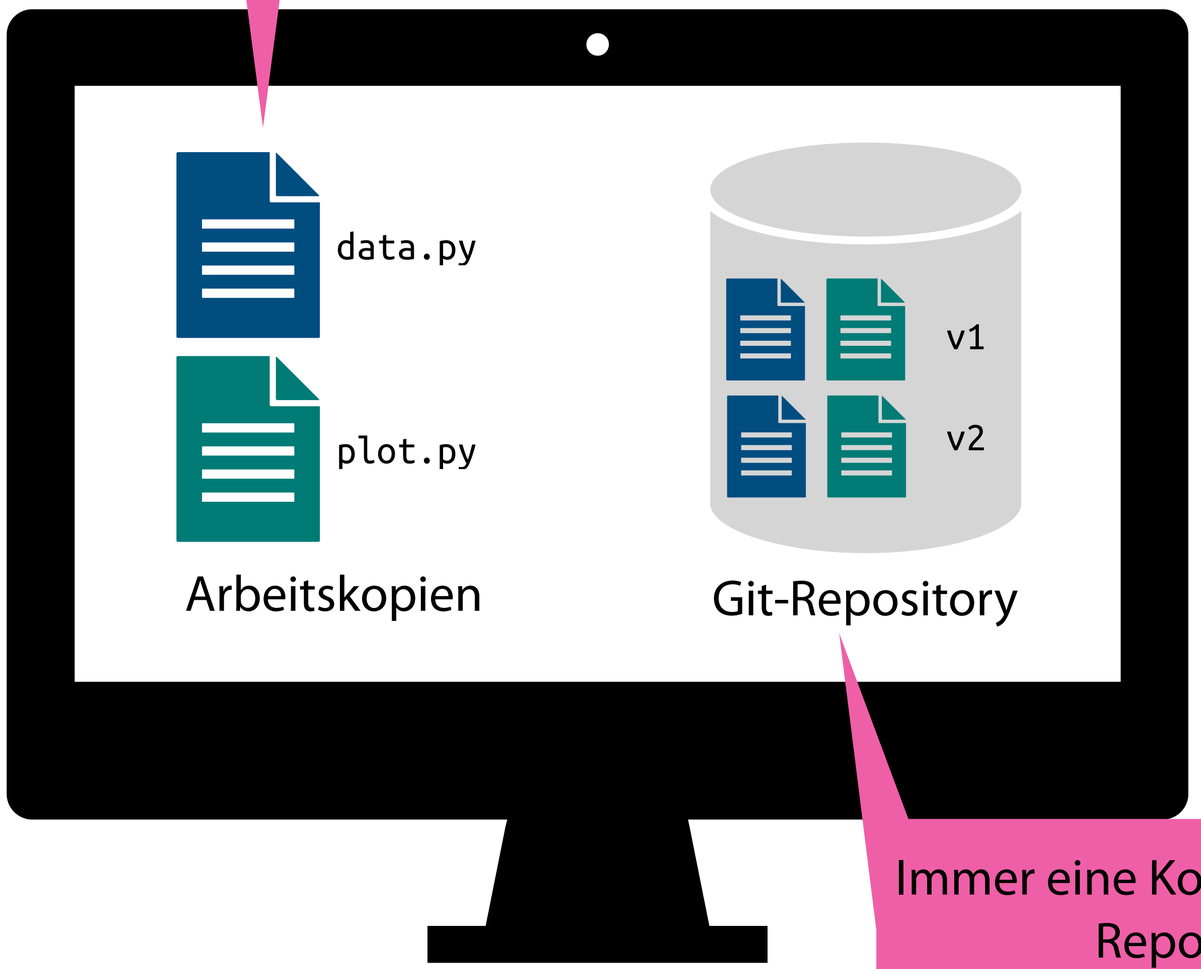
Verteilte Repositories



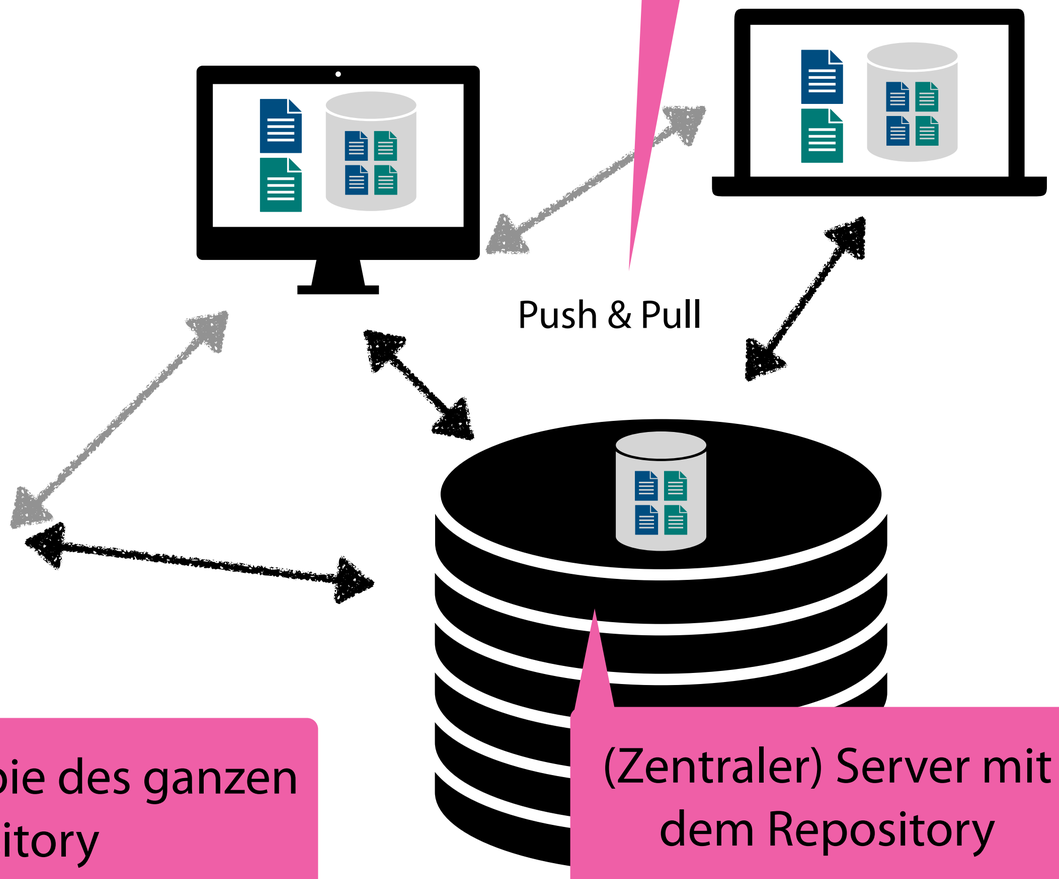


erteilte Repositories

Dateiversion zum Bearbeiten und ins Repository *commiten* oder auch zurücksetzen



Zusammenführen und austauschen der Repository (haupt. zwischen Client & Server)



Immer eine Kopie des ganzen Repository

(Zentraler) Server mit dem Repository

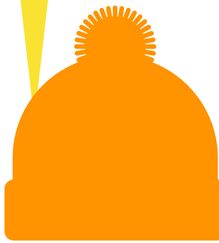
Git

- Mittlerweile Standard (insb. für OpenSource-Software)
  - Entwickelt von Linus Torvalds für Linux-Kernel
- SHA-1 Hash für jede Änderung
- Vollständig lokal nutzbar, kein (zentraler) Server nötig

Git

- Mittlerweile Standard (insb. für OpenSource-Software)
  - Entwickelt von Linus Torvalds für Linux-Kernel
- SHA-1 Hash für jede Änderung
- Vollständig lokal nutzbar, kein (zentraler) Server nötig

Ein Verlust der Daten auf einem Repository-Server lässt sich direkt aus dem lokalen Repository wiederherstellen.



Installation

- Vorinstalliert auf MacOS
- Paketquellen unter Linux
- Download und Anleitung für Windows
  https://git-scm.com/downloads

Installation

- Vorinstalliert auf MacOS
- Paketquellen unter Linux
- Download und Anleitung für Windows https://git-scm.com/downloads
- Push & Pull

Installation

- Vorinstalliert auf MacOS
- Paketquellen unter Linux
- Download und Anleitung für Windows
  https://git-scm.com/downloads

- Push & Pull
  - SSH Authentifikation
    Anleitung von GitHub

Installation

- Vorinstalliert auf MacOS
- Paketquellen unter Linux
- Download und Anleitung für Windows https://git-scm.com/downloads

- Push & Pull
  - SSH Authentifikation Anleitung von GitHub
  - Alternativ mittels Username/ Passwort über HTTP(S)

Konfiguration

```
git config --global user.name "My Name"
git config --global user.email "me@example.com"
```

Konfiguration

Gilt für ganzen Benutzeraccount, ohne `--global` für aktuelles Repo.

```
git config --global user.name "My Name"
git config --global user.email "me@example.com"
```

Konfiguration

Gilt für ganzen Benutzeraccount, ohne `--global` für aktuelles Repo.

```
git config --global user.name "My Name"
git config --global user.email "me@example.com"
```

- Commits (Änderungen Repository) werden damit versehen
  - Muss keine *echte* Mail und auch nicht der *echte* Name sein
- Weitere Konfiguration ist nicht notwendig

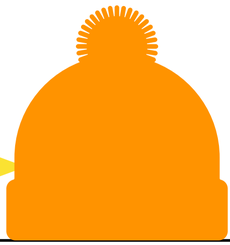
Konfiguration

Gilt für ganzen Benutzeraccount, ohne `--global` für aktuelles Repo.

```
git config --global user.name "My Name"
git config --global user.email "me@example.com"
```

- Commits (Änderungen Repository) werden damit versehen
  - Muss keine *echte* Mail und auch nicht der *echte* Name sein
- Weitere Konfiguration ist nicht notwendig

Tauscht man Commits aus oder pusht sie in ein andere Repository, dann werden Namen & E-Mail geteilt.



Konfiguration

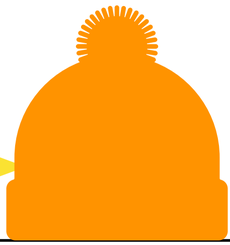
Gilt für ganzen Benutzeraccount, ohne `--global` für aktuelles Repo.

```
git config --global user.name "My Name"
git config --global user.email "me@example.com"
```

- Commits (Änderungen Repository) werden damit versehen
  - Muss keine *echte* Mail und auch nicht der *echte* Name sein
- Weitere Konfiguration ist nicht notwendig

Dann aber keine Zuordnung der Commits möglich, daher eine „öffentliche“ Mail-Adresse nutzen.

Tauscht man Commits aus oder pusht sie in ein andere Repository, dann werden Namen & E-Mail geteilt.



II.
Git

2. Lokal: Commit, Stash, Branch, Merge

Das erste Repository

```
$> git init
  Initialized empty Git repository in ./.git/

$> tree -a .
.
└── .git
    ├── HEAD
    ├── config
    ├── description
    ├── hooks
    │   ├── …
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags
```

Das erste Repository

```
$> git init
  Initialized empty Git repository in ./.git/

$> tree -a .
.
└── .git
    ├── HEAD
    ├── config
    ├── description
    ├── hooks
    │   ├── …
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags
```

data.py

plot.py

Arbeitskopien

Das erste Repository

```
$> git init
  Initialized empty Git repository in ./.git/

$> tree -a .
.
└── .git
    ├── HEAD
    ├── config
    ├── description
    ├── hooks
    │   ├── …
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags
```

data.py

plot.py

Arbeitskopien

Git-Repository

Das erste Repository

```
$> git init
  Initialized empty Git repository in ./.git/

$> tree -a .
.
└── .git
    ├── HEAD
    ├── config
    ├── description
    ├── hooks
    │   ├── …
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags
```


Git-Repository

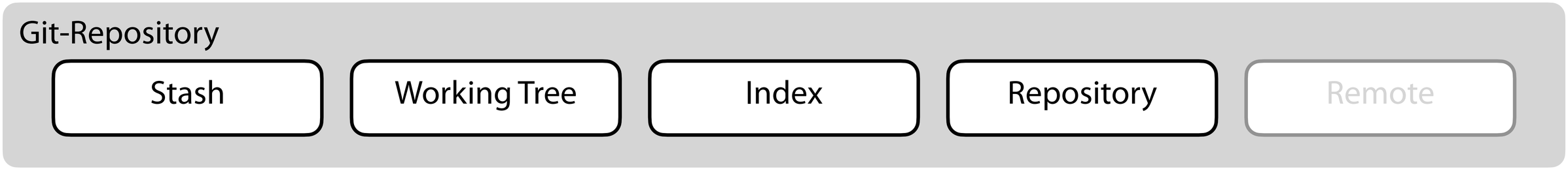
data.py

plot.py

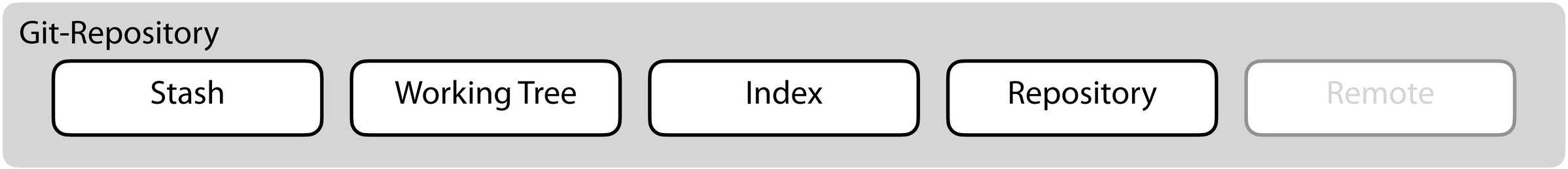
Arbeitskopien

- Neues leeres Repo `./.git/`
- Arbeitskopien `./`

Dateien hinzufügen

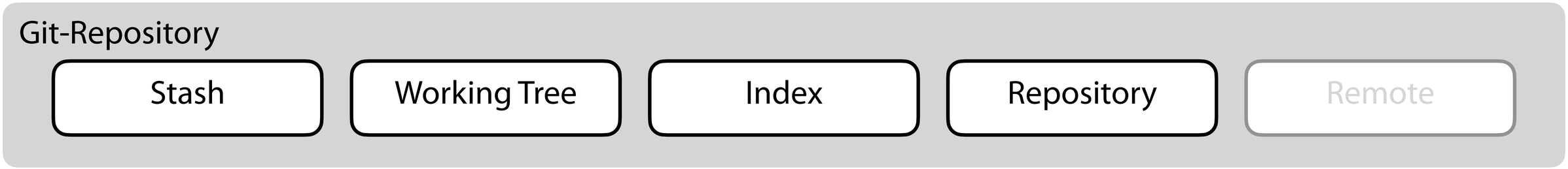


Dateien hinzufügen




```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

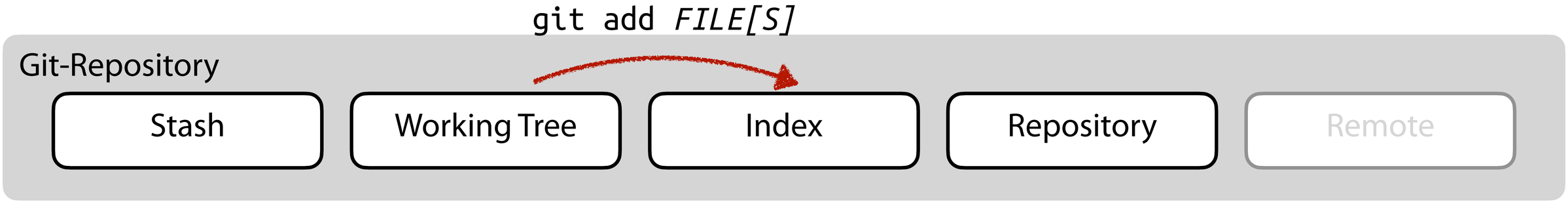
Dateien hinzufügen




```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

```
$> git status
On branch main
  No commits yet
  Untracked files:
    a.txt
$> git add .
```

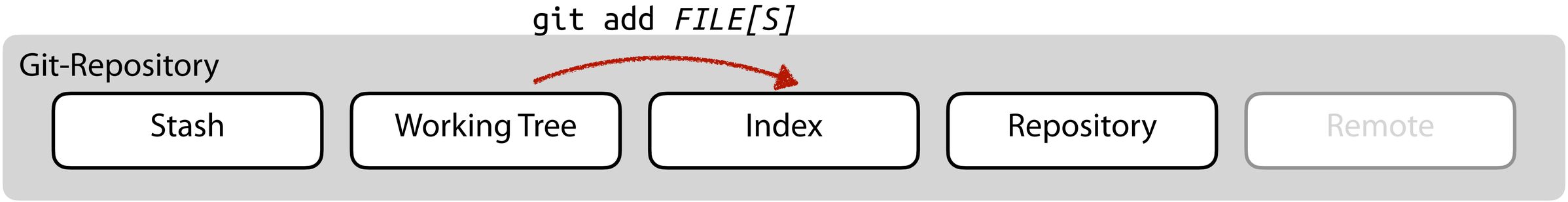
Dateien hinzufügen




```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

```
$> git status
On branch main
  No commits yet
  Untracked files:
    a.txt
$> git add .
```

Dateien hinzufügen

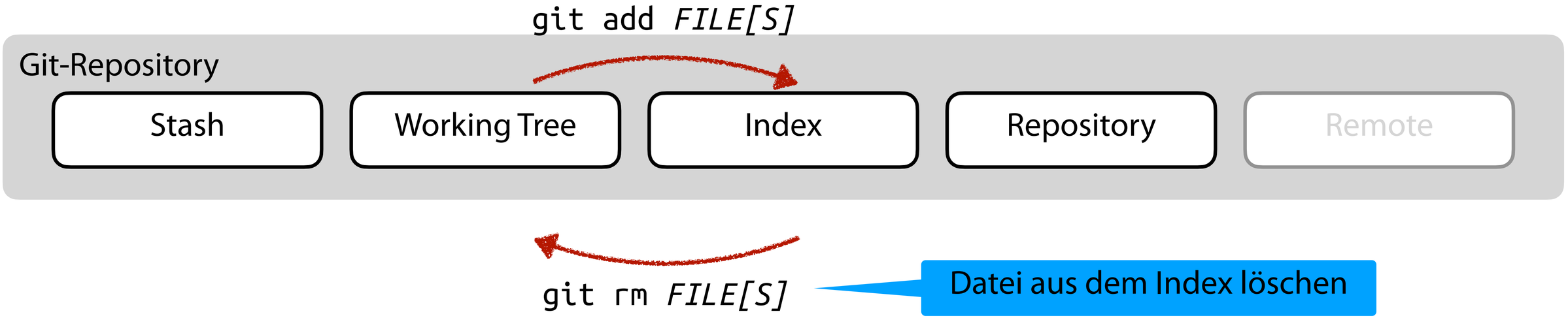



```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

```
$> git status
On branch main
  No commits yet
  Untracked files:
    a.txt
$> git add .
```

```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
```

Dateien hinzufügen

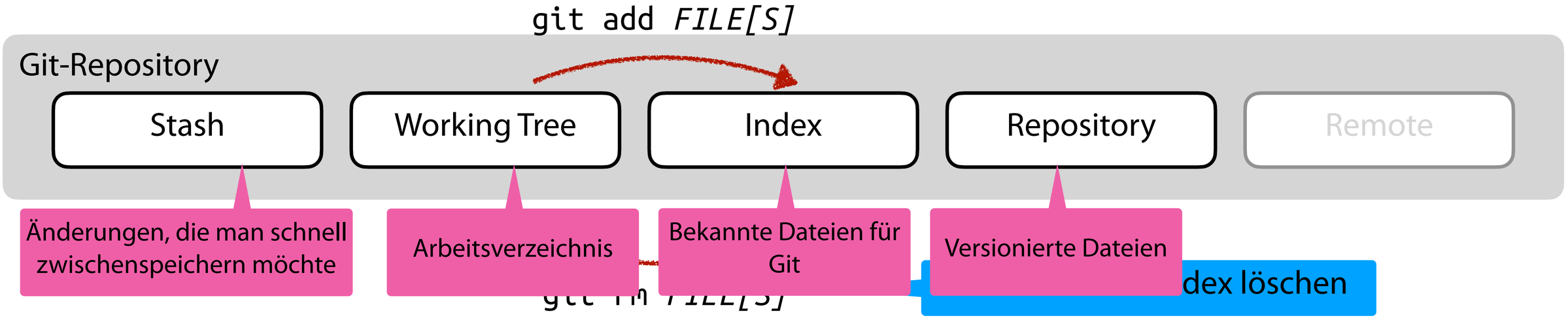



```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

```
$> git status
On branch main
  No commits yet
  Untracked files:
    a.txt
$> git add .
```

```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
```

Dateien hinzufügen

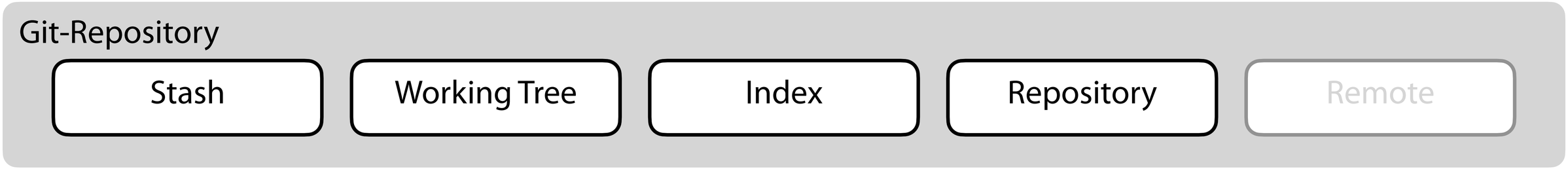



```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

```
$> git status
On branch main
  No commits yet
  Untracked files:
    a.txt
$> git add .
```

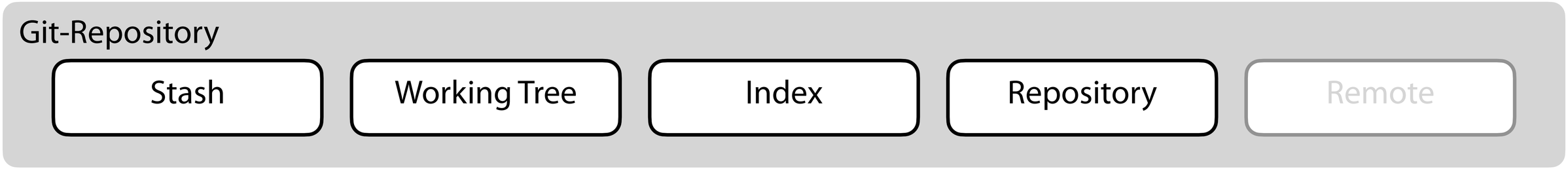
```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
```

Commit




```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
    new file:   b.txt
```

Commit

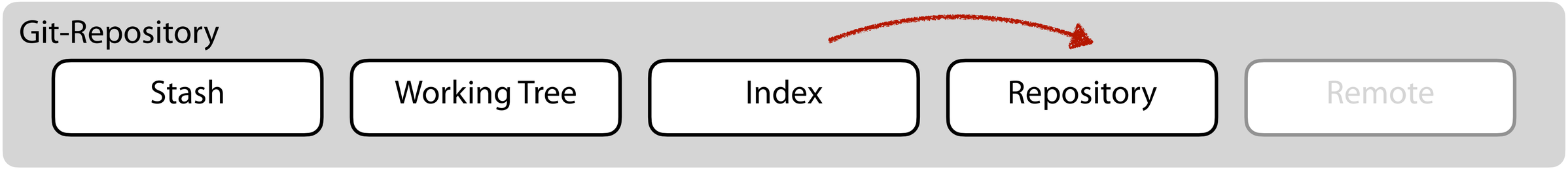



```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
    new file:   b.txt
```

```
$> git commit -m "Meine Änderung"
  [main (root-commit) 2655955] Meine Änderung
    2 files changed, 0 insertions(+), 0 deletions(-)
    create mode 100644 a.txt
    create mode 100644 b.txt
$> git status
  On branch main
    nothing to commit, working tree clean
```

Commit

`git commit [-m "Meine Änderung"]`

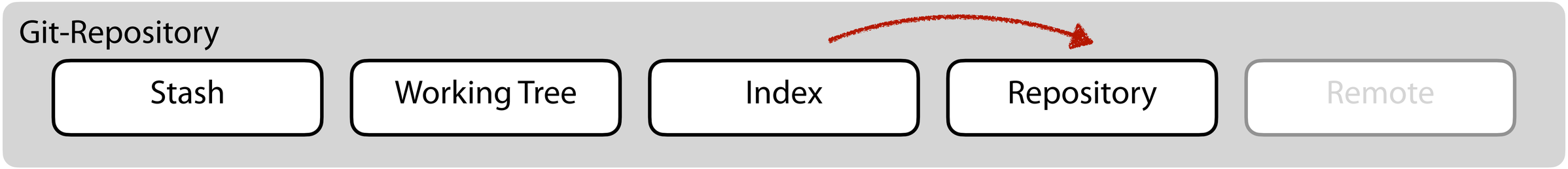



```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
    new file:   b.txt
```

```
$> git commit -m "Meine Änderung"
  [main (root-commit) 2655955] Meine Änderung
    2 files changed, 0 insertions(+), 0 deletions(-)
    create mode 100644 a.txt
    create mode 100644 b.txt
$> git status
  On branch main
    nothing to commit, working tree clean
```

Commit

`git commit [-m "Meine Änderung"]`



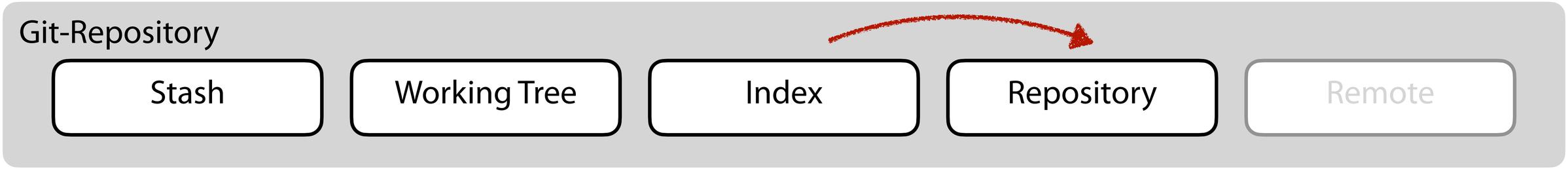
Kurz beschreiben, was getan wurde
(ohne `-m` öffnet sich ein Editor für mehr Text)

```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
    new file:   b.txt
```

```
$> git commit -m "Meine Änderung"
  [main (root-commit) 2655955] Meine Änderung
    2 files changed, 0 insertions(+), 0 deletions(-)
    create mode 100644 a.txt
    create mode 100644 b.txt
$> git status
  On branch main
    nothing to commit, working tree clean
```

Commit

`git commit [-m "Meine Änderung"]`

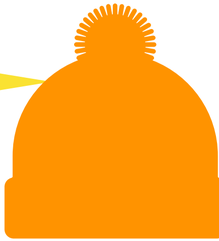


Kurz beschreiben, was getan wurde
(ohne `-m` öffnet sich ein Editor für mehr Text)

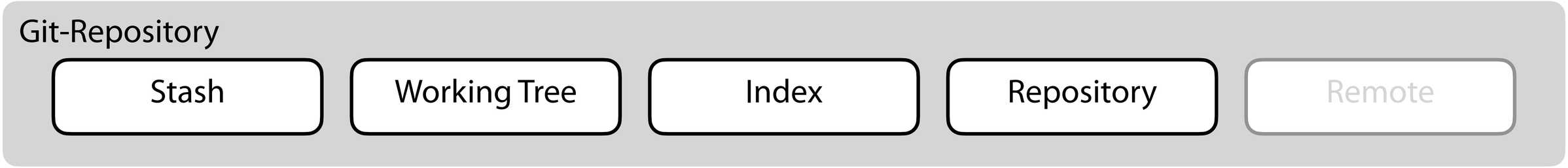
```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
    new file:   b.txt
```

```
$> git commit -m "Meine Änderung"
  [main (root-commit) 2655955] Meine Änderung
    2 files changed, 0 insertions(+), 0 deletions(-)
    create mode 100644 a.txt
    create mode 100644 b.txt
$> git status
  On branch main
    nothing to commit, w
```

Man löscht keine Commits, denn andere könnten die bereits haben und darauf aufbauen.
Besser, neuer Commit mit Änderungen!



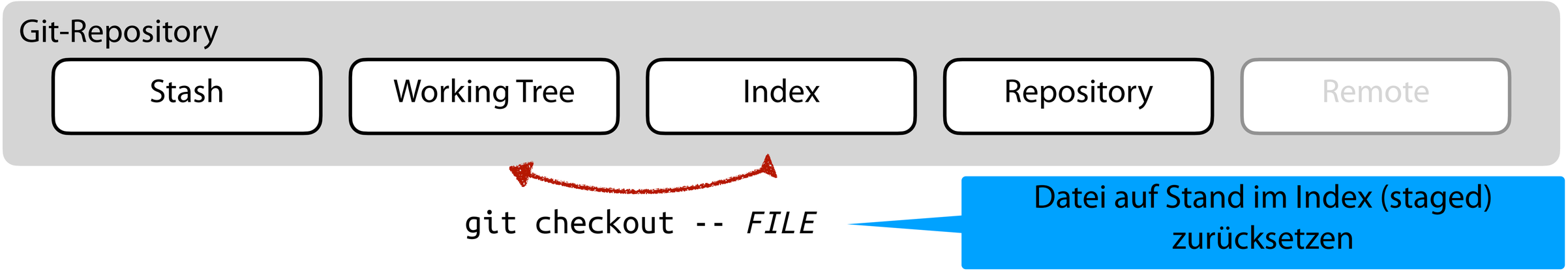
Weitere Änderungen




```
$> git status
On branch main
Changes not staged for commit:
  modified:   a.txt
$> git add .
$> vim b.txt
```

```
$> git status
  On branch main
  Changes to be committed:
    modified:   a.txt
  Changes not staged for commit:
    modified:   b.txt
```

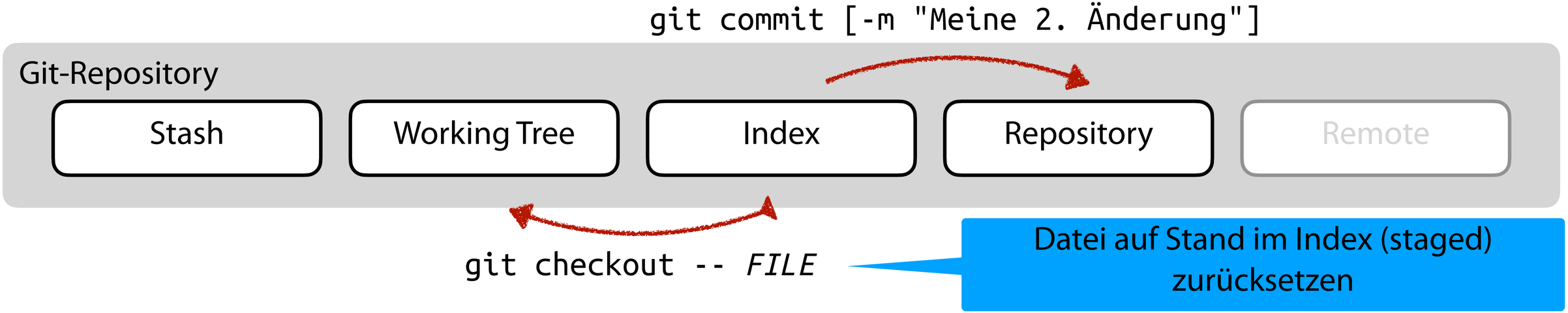
Weitere Änderungen




```
$> git status
On branch main
Changes not staged for commit:
  modified:   a.txt
$> git add .
$> vim b.txt
```

```
$> git status
  On branch main
  Changes to be committed:
    modified:   a.txt
  Changes not staged for commit:
    modified:   b.txt
```

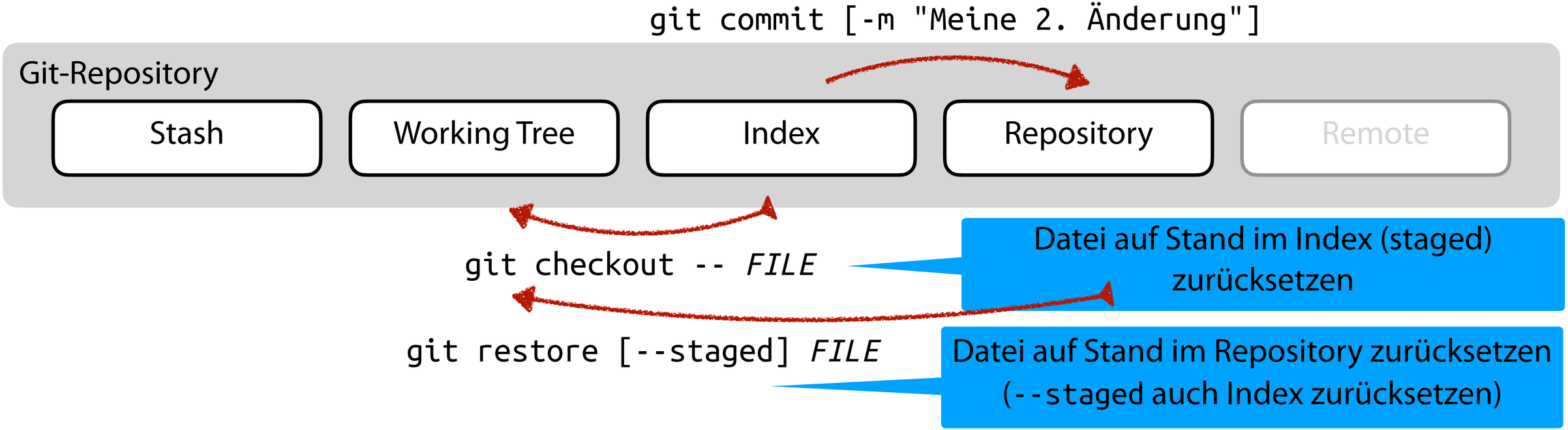
Weitere Änderungen




```
$> git status
On branch main
Changes not staged for commit:
  modified:   a.txt
$> git add .
$> vim b.txt
```

```
$> git status
  On branch main
  Changes to be committed:
    modified:   a.txt
  Changes not staged for commit:
    modified:   b.txt
```

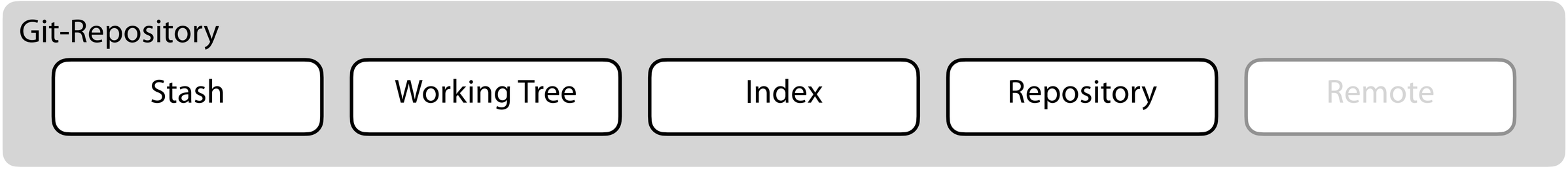
Weitere Änderungen



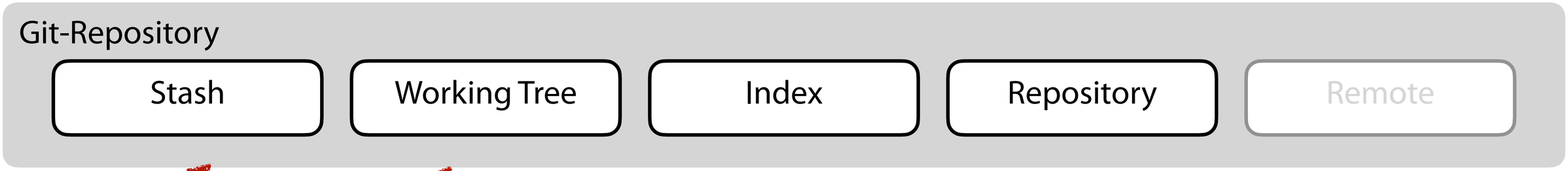

```
$> git status
On branch main
Changes not staged for commit:
  modified:   a.txt
$> git add .
$> vim b.txt
```

```
$> git status
  On branch main
  Changes to be committed:
    modified:   a.txt
  Changes not staged for commit:
    modified:   b.txt
```

Zwischenspeicher



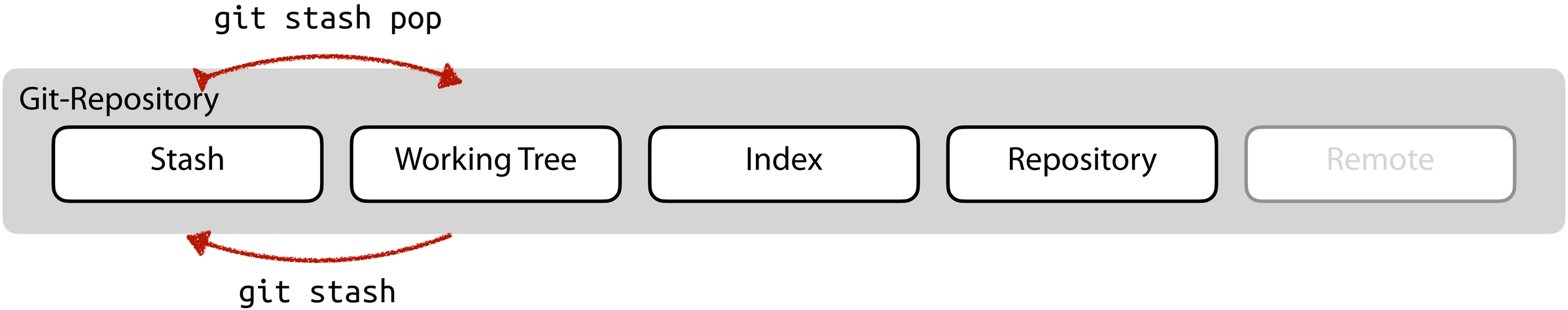
Zwischenspeicher



`git stash`

```
$> git status
  On branch main
  Changes to be committed:
    modified:   a.txt
$> git stash
  Saved working directory …
$> git status
  On branch main
  nothing to commit, working tree clean
```

Zwischenspeicher




```
$> git status
  On branch main
  Changes to be committed:
    modified:   a.txt
$> git stash
  Saved working directory …
$> git status
  On branch main
  nothing to commit, working tree clean
```

```
$> git stash pop
  On branch main
  Changes not staged for commit:
    modified:   a.txt
  no changes added to commit
  Dropped refs/stash@{0}
    (0e71d0d32adcbf16fbe6a10c1f27436012d7f726)
```

Verlauf der Commits

```
$> git log --oneline
```

Verlauf der Commits

```
$> git log --oneline
```

```
d1a8079 (HEAD -> master, tag: v2.3.9, GH/master, GH/HEAD) UnRead does not use Podcast ID any more, s
o moving podcast does not remove all "Reads"
15c4a5c Update VERSION
63b0336 Merge pull request #9 from DirkBaumeister/feature/added-configurable-setupapp-ident
f92e070 fixed typo in readme file
8752f21 added description of environment variable to readme
2ea61d8 fixed typo in word ident
1bda2ae added configurable setuapp ident string
aa95130 (tag: v2.3.7) Add Preview
90beeb0 Chnage year in copyright
2db9bac (tag: v2.3.6) Optomize index.php exec. order
aeecd70 (tag: v2.3.5) Fix sorting bug
4b2140d (tag: v2.3.4) Sort stations and podcasts by name
ebbd82b Current Dist & Docker Fix
4d07b1a Update Docker
b02209c (tag: v2.3.3) Fix PHP-8 Bug Stream.php
fe888ef (tag: v2.3.2) Code Check und Docker PHP-8
f5e56dd (tag: v2.3.1) Update jQuery
83dfa19 Update startup-before.sh
425d997 (tag: v2.3.0) Manage Un/Read in GUI Preview
:
```

q um zu schließen

Verlauf der Commits

```
$> git log --oneline
```

```
d1a8079 (HEAD -> master, tag: v2.3.9, GH/master, GH/HEAD) UnRead does not use Podcast ID any more, s
o moving podcast does not remove all "Reads"
15c4a5c Update VERSION
63b0336 Merge pull request #9 from DirkBaumeister/feature/added-configurable-setupapp-ident
f92e070 fixed typo in readme file
8752f21 added description of environment variable to readme
2ea61d8 fixed typo in word ident
1bda2ae added configurable setuapp ident string
aa95130 (tag: v2.3.7) Add Preview
90beeb0 Chnage year in copyright
2db9bac (tag: v2.3.6) Optomize index.php exec. order
aeecd70 (tag: v2.3.5) Fix sorting bug
4b2140d (tag: v2.3.4) Sort stations and podcasts by name
ebbd82b Current Dist & Docker Fix
4d07b1a Update Docker
b02209c (tag: v2.3.3) Fix PHP-8 Bug Stream.php
fe888ef (tag: v2.3.2) Code Check und Docker PHP-8
f5e56dd (tag: v2.3.1) Update jQuery
83dfa19 Update startup-before.sh
425d997 (tag: v2.3.0) Manage Un/Read in GUI Preview
:
```

`git checkout` *HASH* ändert Working Tree auf Stand des Commits

q um zu schließen

Verlauf der Commits

```
$> git log --oneline
```

```
d1a8079 (HEAD -> master, tag: v2.3.9, GH/master, GH/HEAD) UnRead does not use Podcast ID any more, s
o moving podcast does not remove all "Reads"
15c4a5c Update VERSION
63b0336 Merge pull request #9 from DirkBaumeister/feature/added-configurable-setupapp-ident
f92e070 fixed typo in readme file
8752f21 added description of environment variable to readme
2ea61d8 fixed typo in word ident
1bda2ae added configurable setuapp ident string
aa95130 (tag: v2.3.7) Add Preview
90beeb0 Chnage year in copyright
2db9bac (tag: v2.3.6) Optomize index.php exec. order
aeecd70 (tag: v2.3.5) Fix sorting bug
4b2140d (tag: v2.3.4) Sort stations and podcasts by name
ebbd82b Current Dist & Docker Fix
4d07b1a Update Docker
b02209c (tag: v2.3.3) Fix PHP-8 Bug Stream.php
fe888ef (tag: v2.3.2) Code Check und Docker PHP-8
f5e56dd (tag: v2.3.1) Update jQuery
83dfa19 Update startup-before.sh
425d997 (tag: v2.3.0) Manage Un/Read in GUI Preview
:
```

`git checkout` *HASH* ändert Working Tree auf Stand des Commits

Jedoch dort keine Änderungen möglich

Zurück mittels `git checkout HEAD`

q um zu schließen

Anforderungen Versionsverwaltung

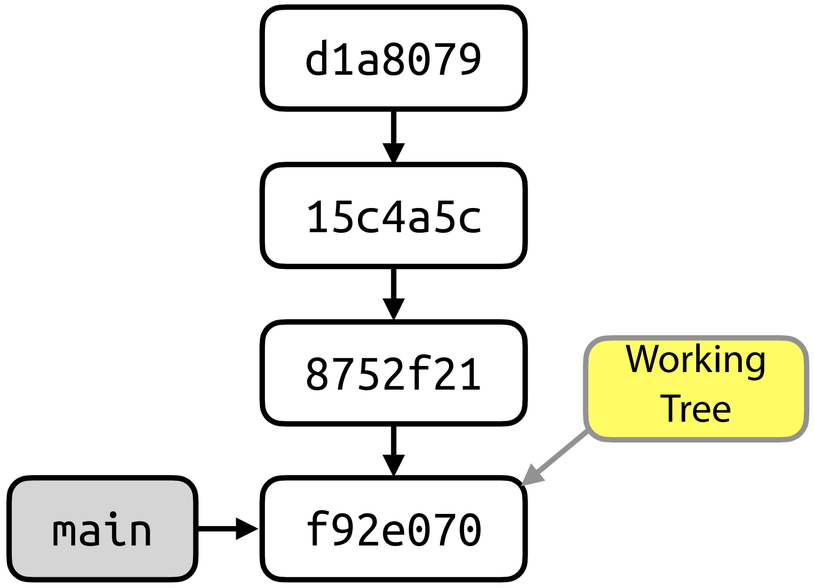
- [x] Verlauf der Änderungen (textbasierte Dateien)
- [ ] Verschiedene **Entwicklungszweige** gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte
- [ ] Zusammenführen von **Entwicklungszweigen**
- [x] Ein (zentrales) Repository

Branches

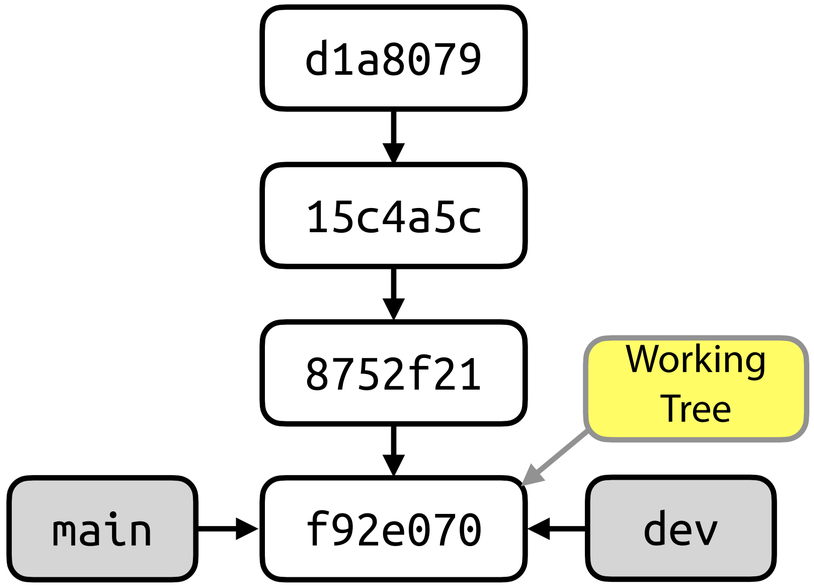
```
* d1a8079 (HEAD -> master, tag: v2.3.9, GH/master, GH/HEAD) UnRead does not use Podcast ID any more,
 so moving podcast does not remove all "Reads"
* 15c4a5c Update VERSION
*   63b0336 Merge pull request #9 from DirkBaumeister/feature/added-configurable-setupapp-ident
|\
| * f92e070 fixed typo in readme file
| * 8752f21 added description of environment variable to readme
| * 2ea61d8 fixed typo in word ident
| * 1bda2ae added configurable setuapp ident string
|/
* aa95130 (tag: v2.3.7) Add Preview
:
```

- Bisher auf Branch `main` (früher `master`)
- Verzweigen der Entwicklung und anschließend vereinen

Branches: Idee

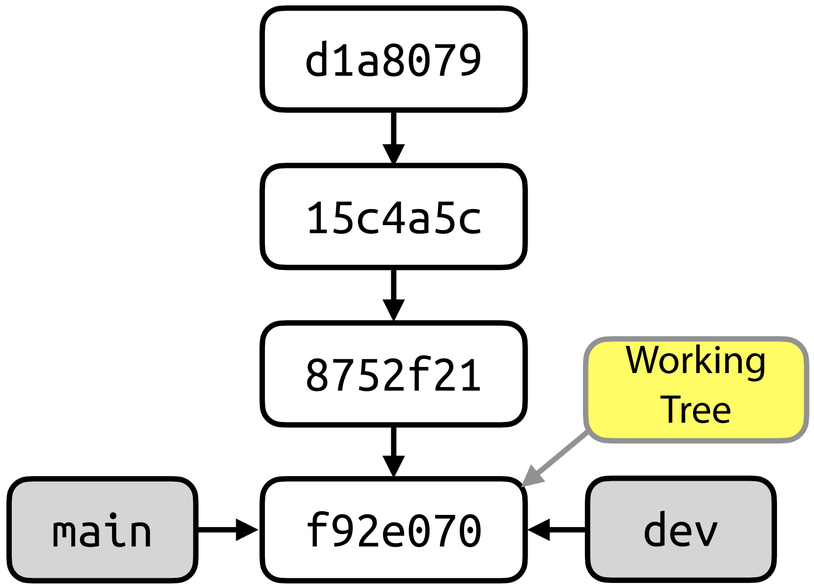


Branches: Idee

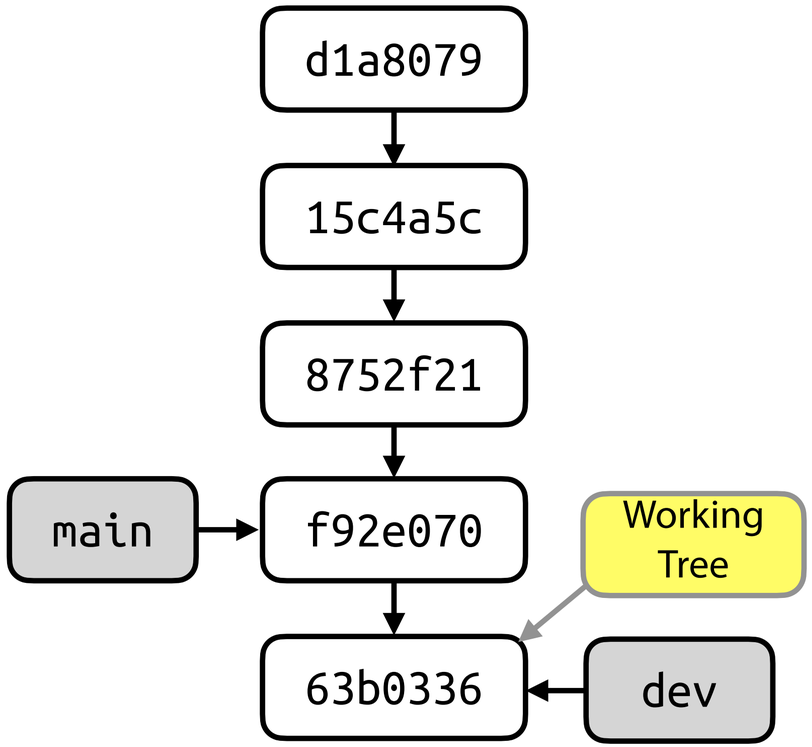



```
$> git branch dev
```

Branches: Idee

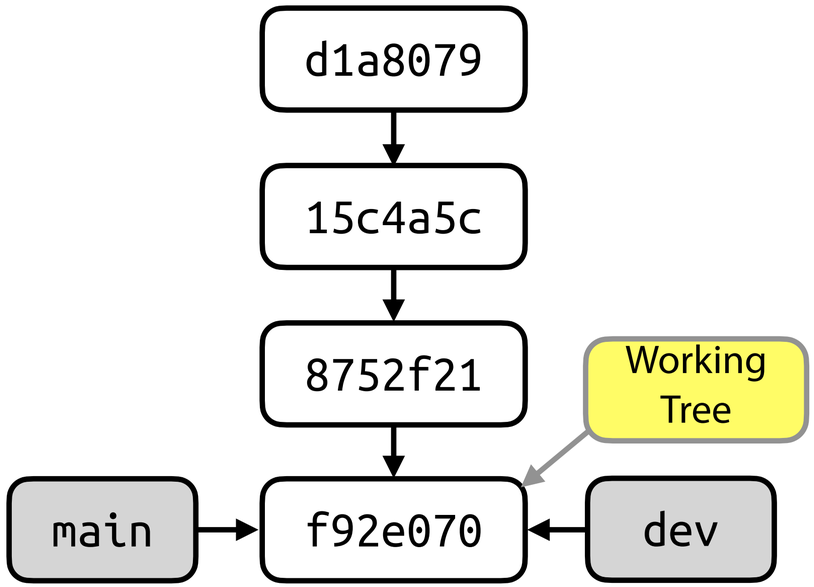



```
$> git branch dev
```

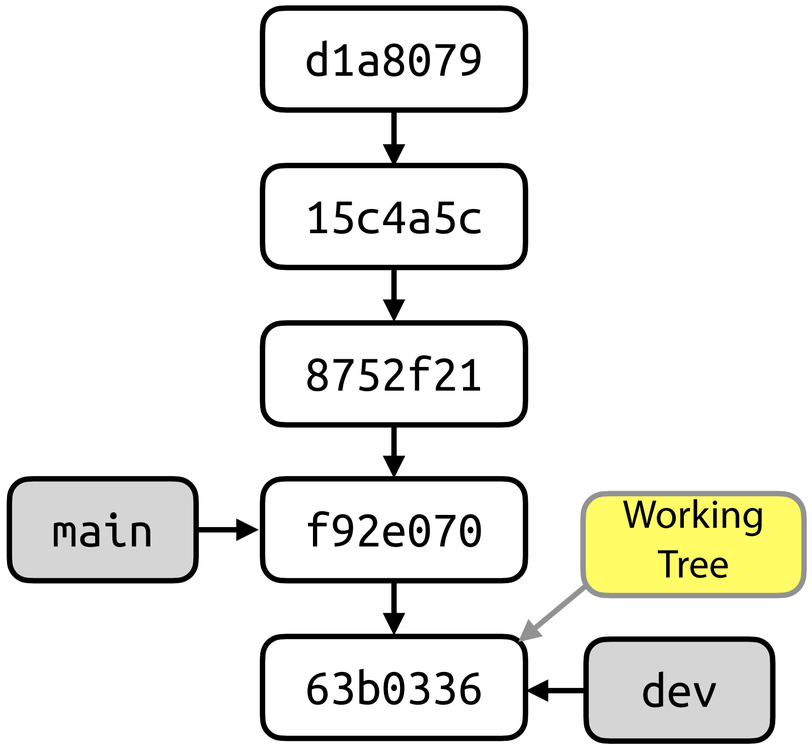



```
$> git checkout dev
$> git commit
```

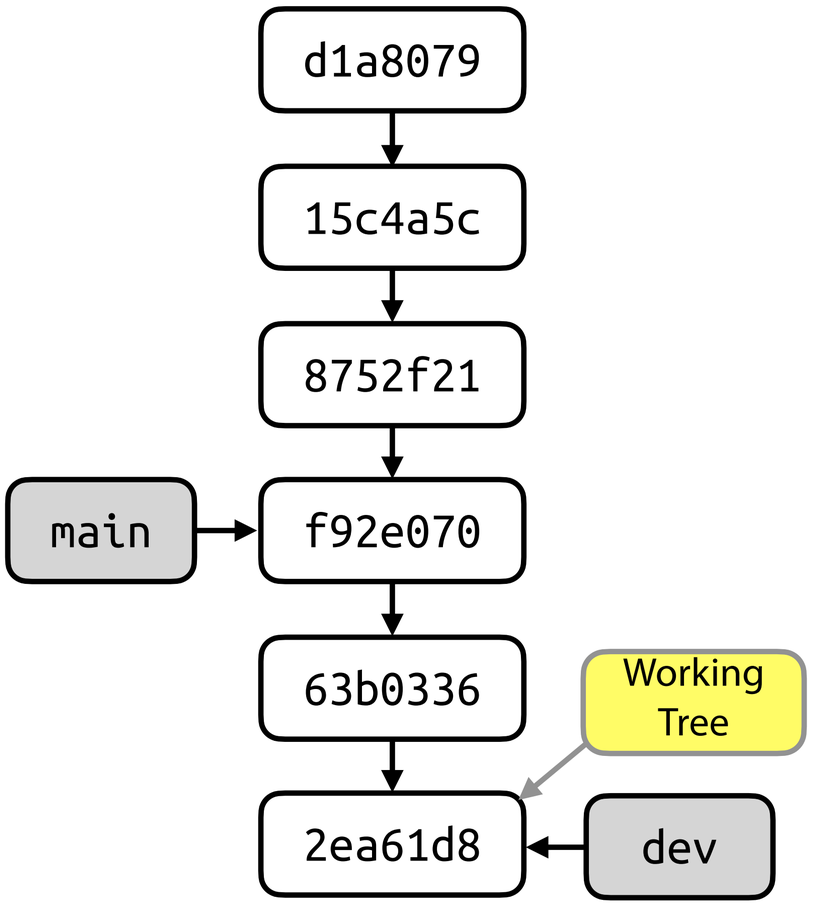
Branches: Idee




```
$> git branch dev
```

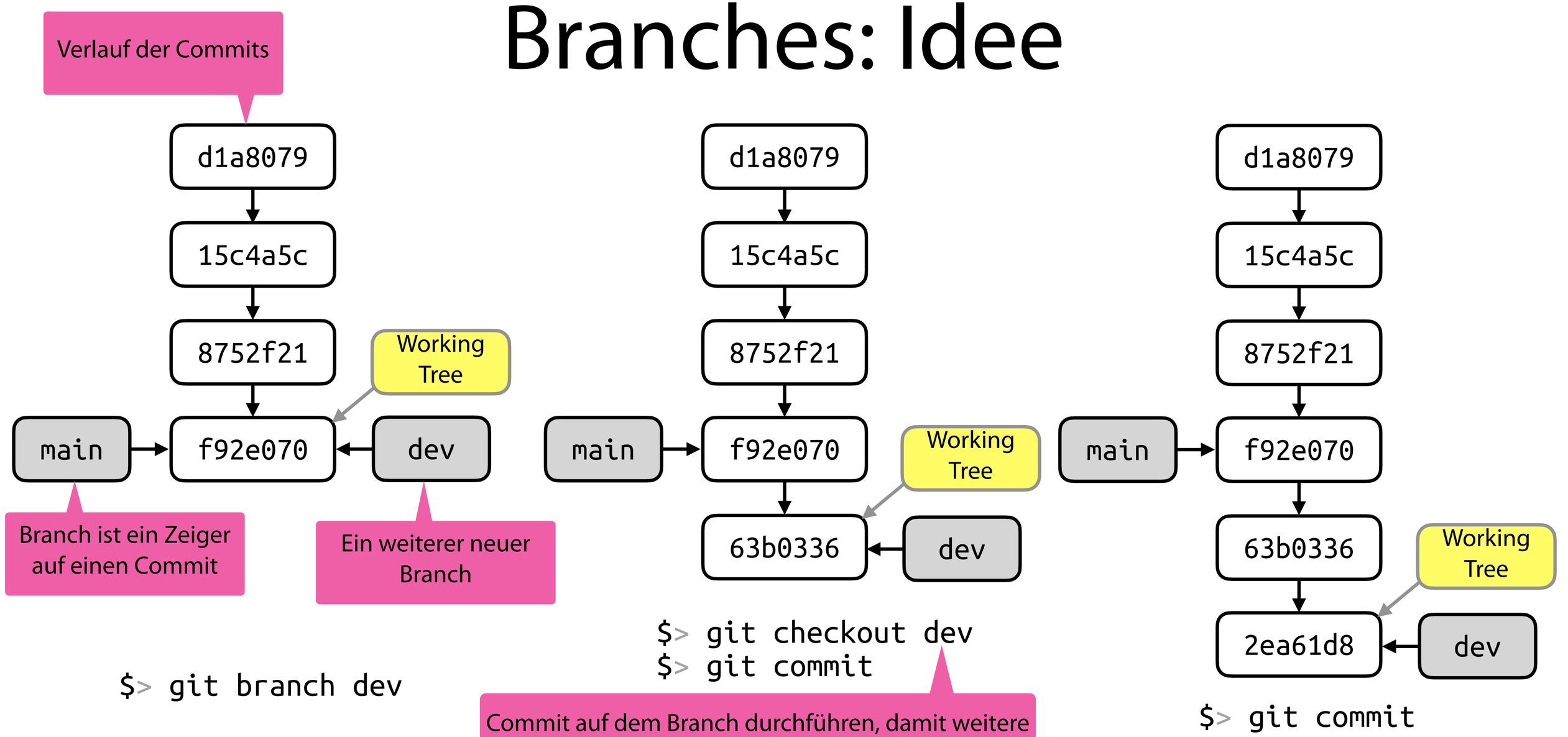



```
$> git checkout dev
$> git commit
```

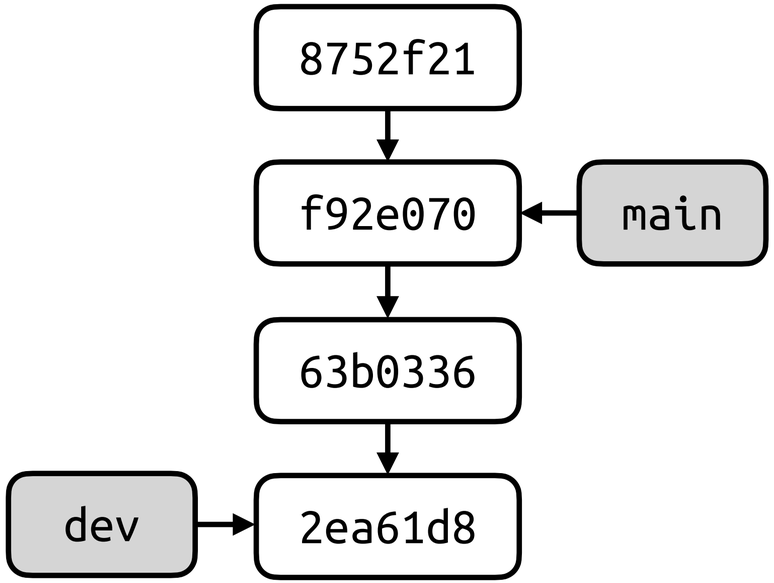



```
$> git commit
```

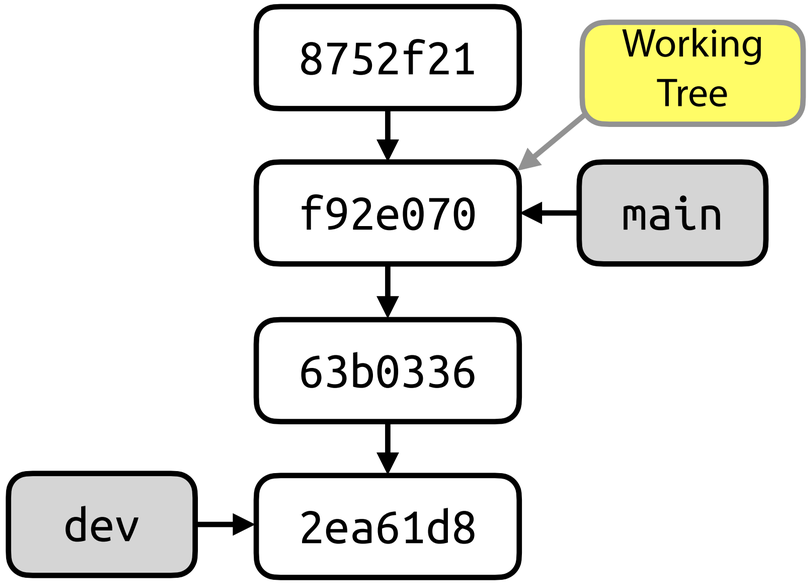
Branches: Idee



Branches: Verzweigung

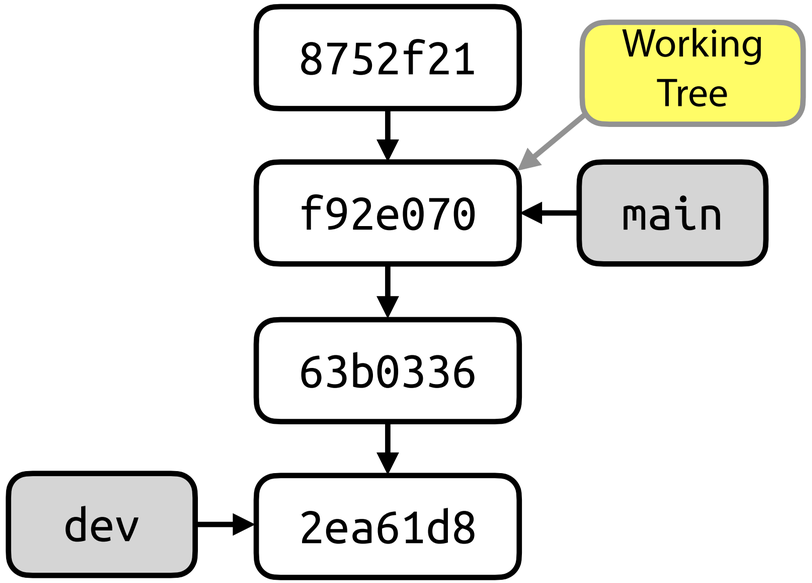


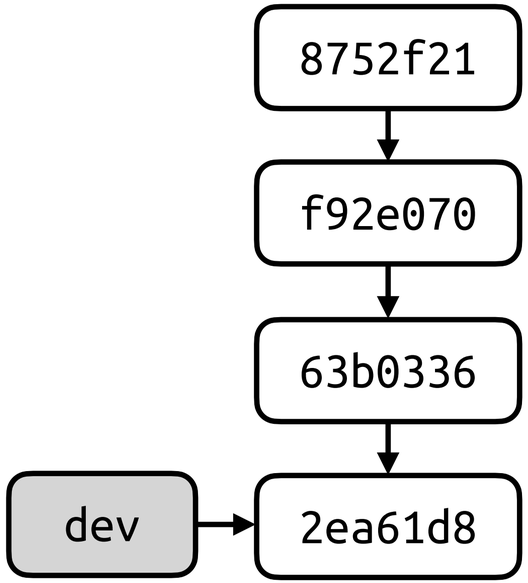
Branches: Verzweigung




```
$> git checkout main
```

Branches: Verzweigung



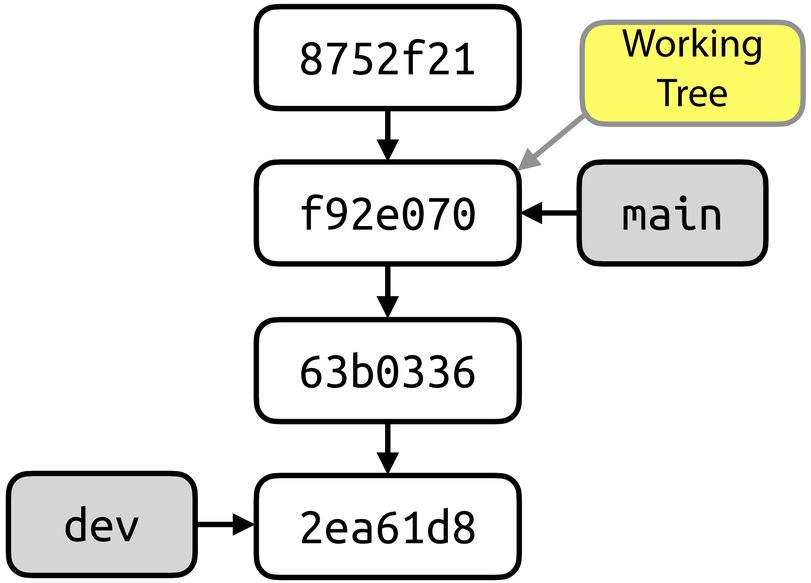


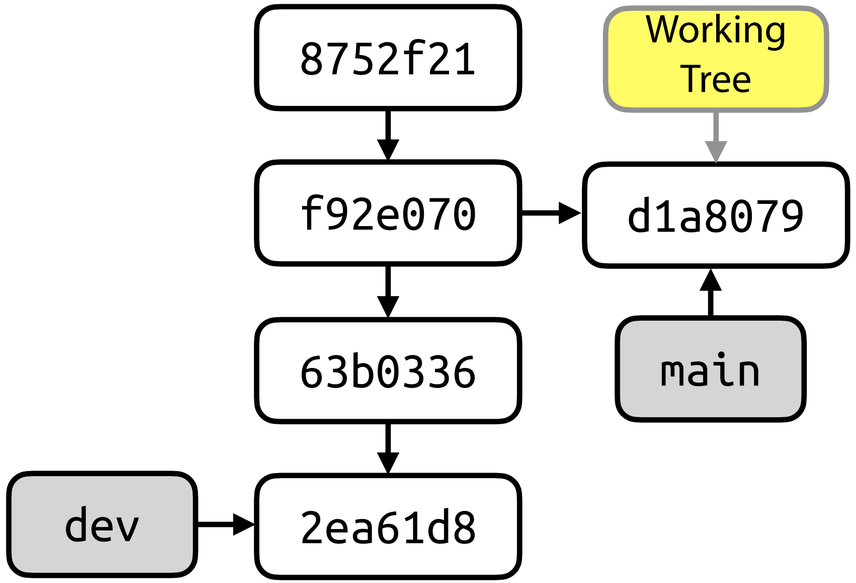

```
$> git checkout main
```

```
$> git commit
```

Was passiert jetzt?

Branches: Verzweigung



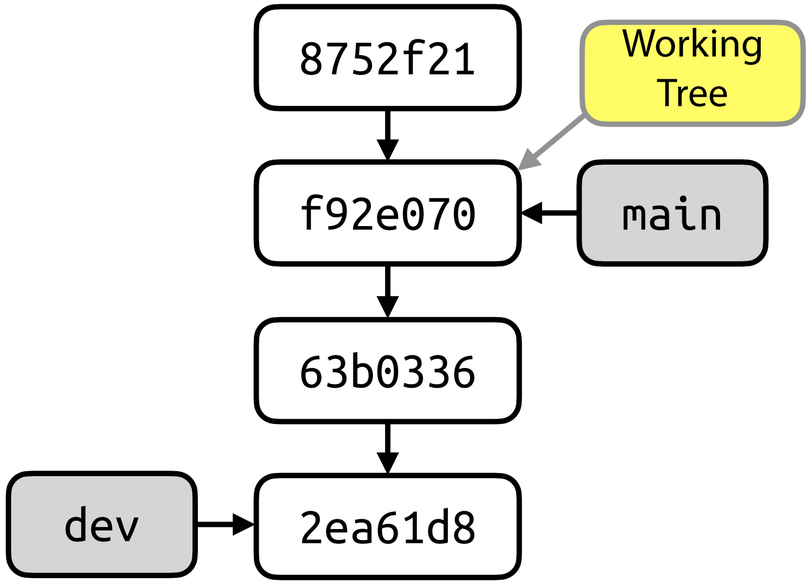



```
$> git checkout main
```

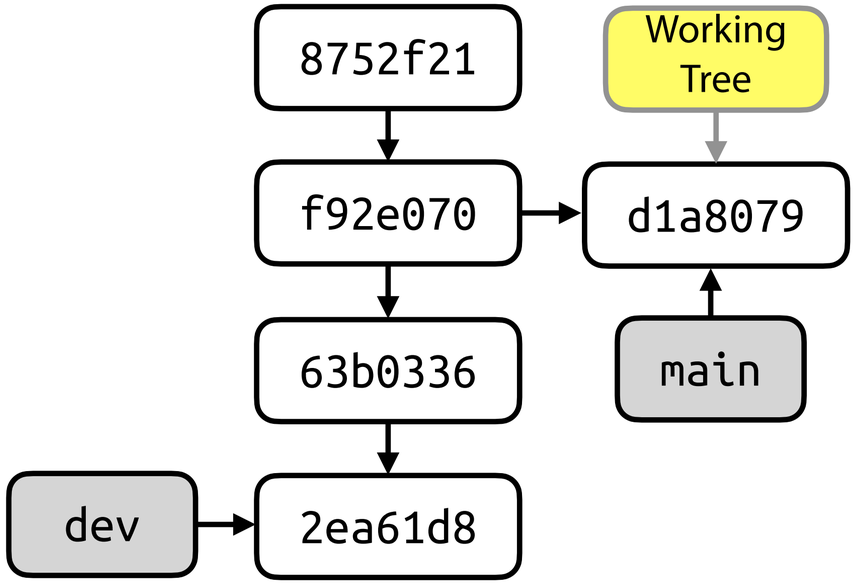
```
$> git commit
```

Was passiert jetzt?

Branches: Verzweigung

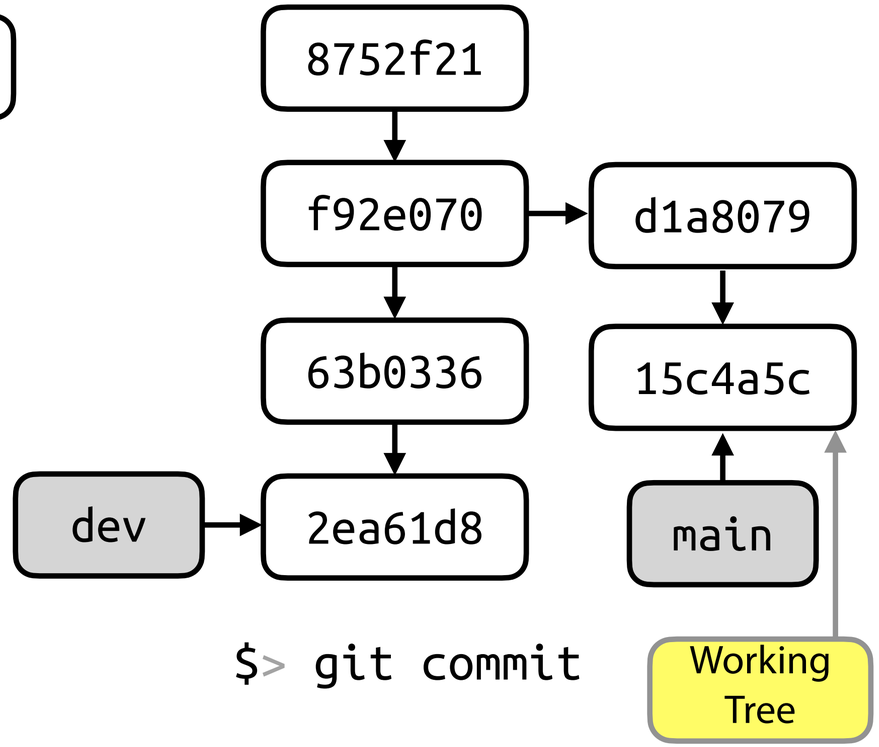



```
$> git checkout main
```



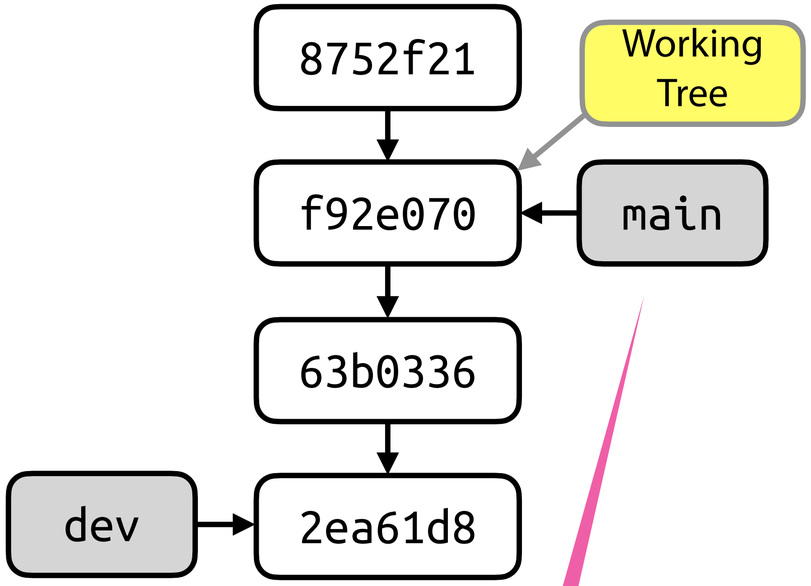

```
$> git commit
```

Was passiert jetzt?



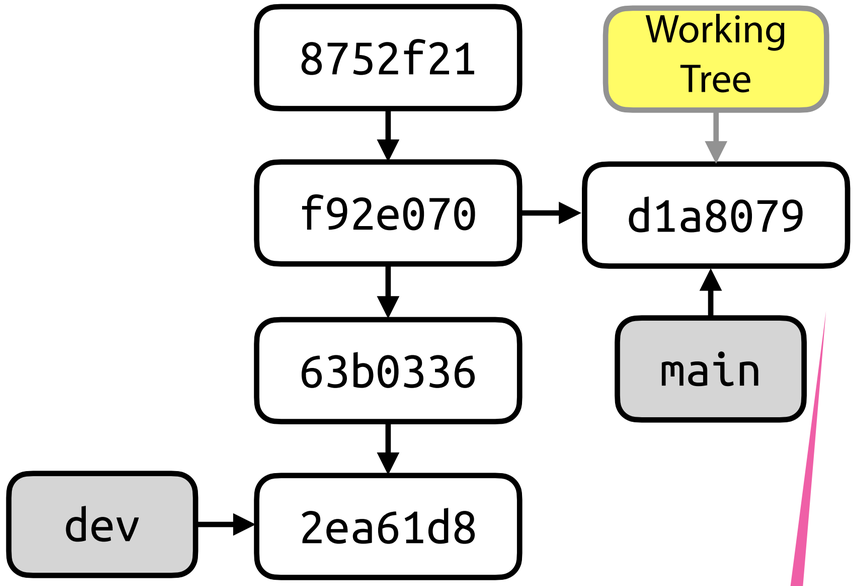

```
$> git commit
```

Branches: Verzweigung




```
$> git checkout main
```

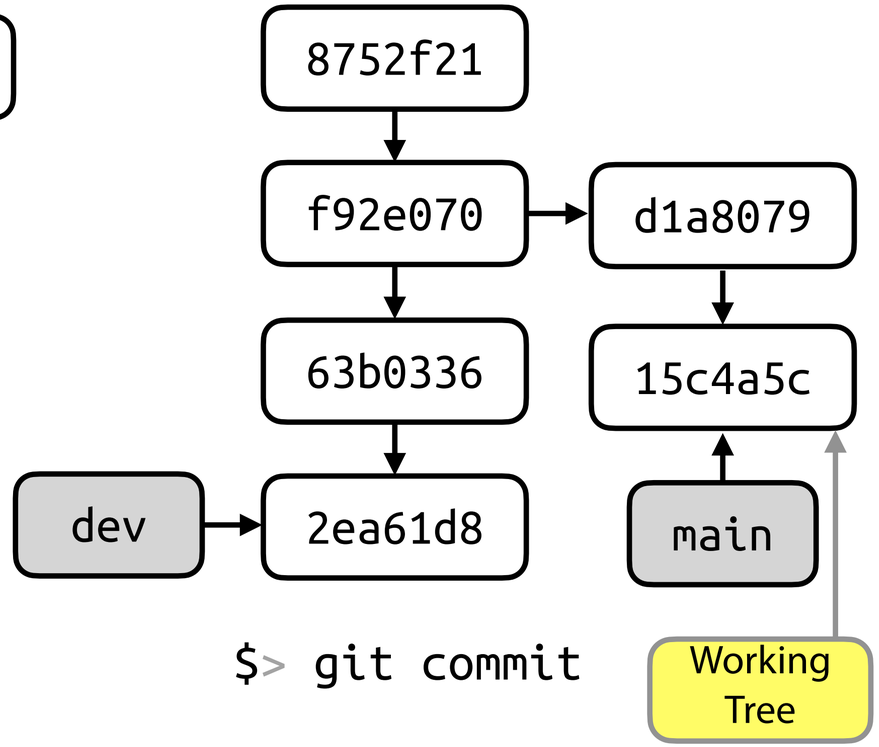
Auf einen anderen Branch wechseln




```
$> git commit
```

Was passiert jetzt?

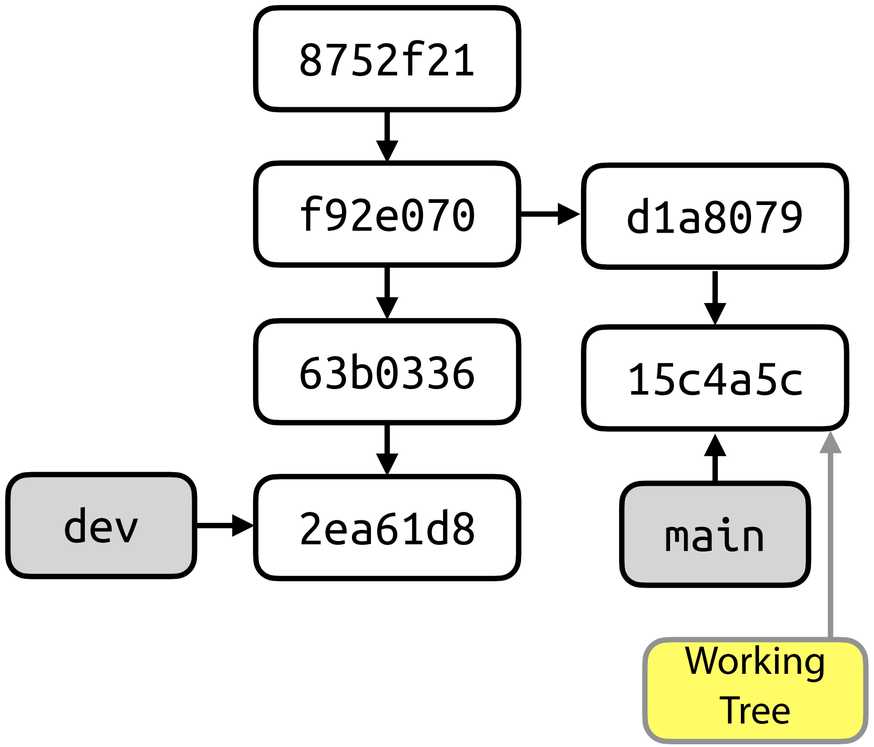
Auf dem anderen Branch Commits hinzufügen



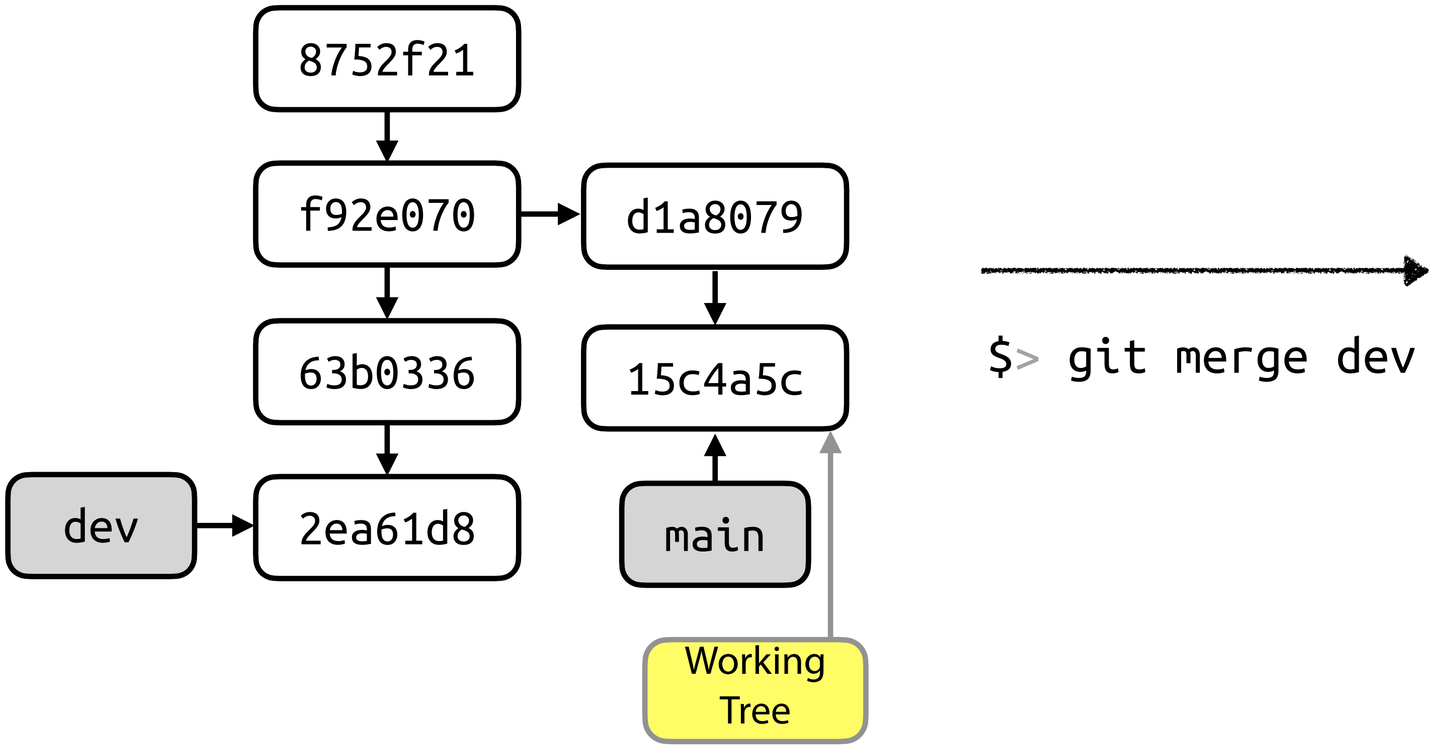

```
$> git commit
```

Working Tree

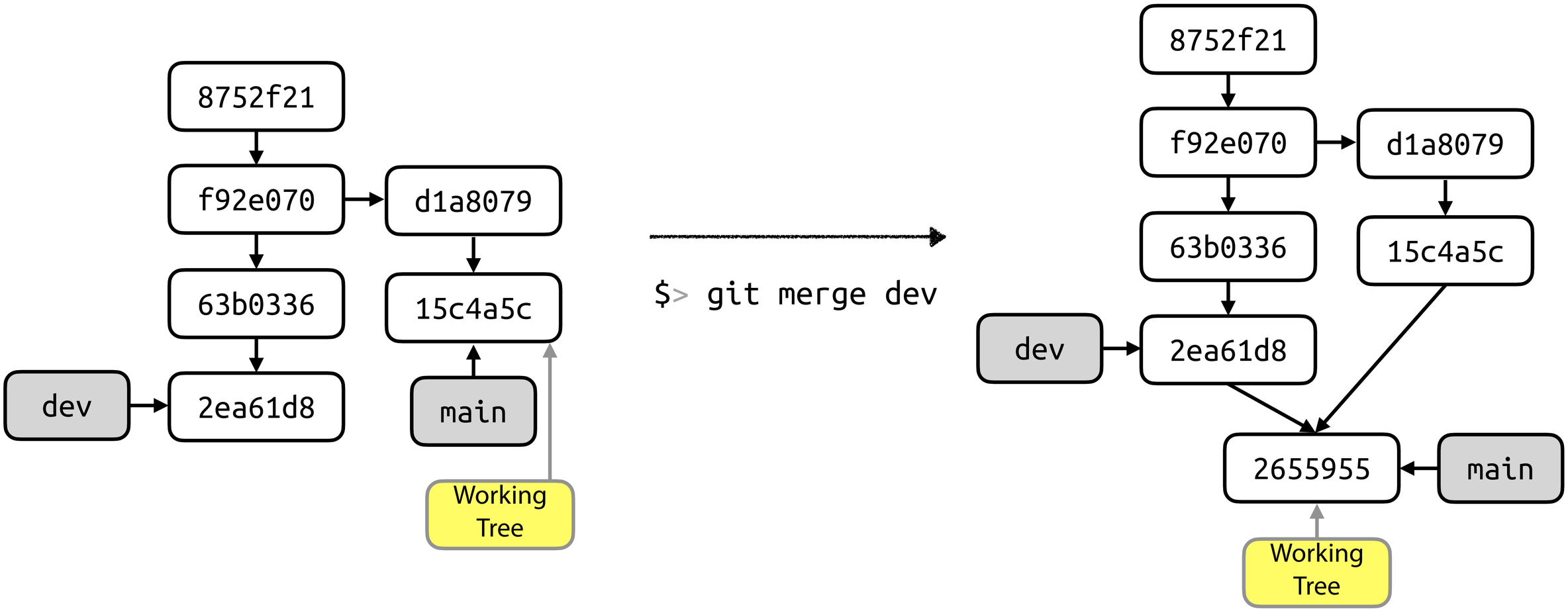
Branches: Zusammenführung



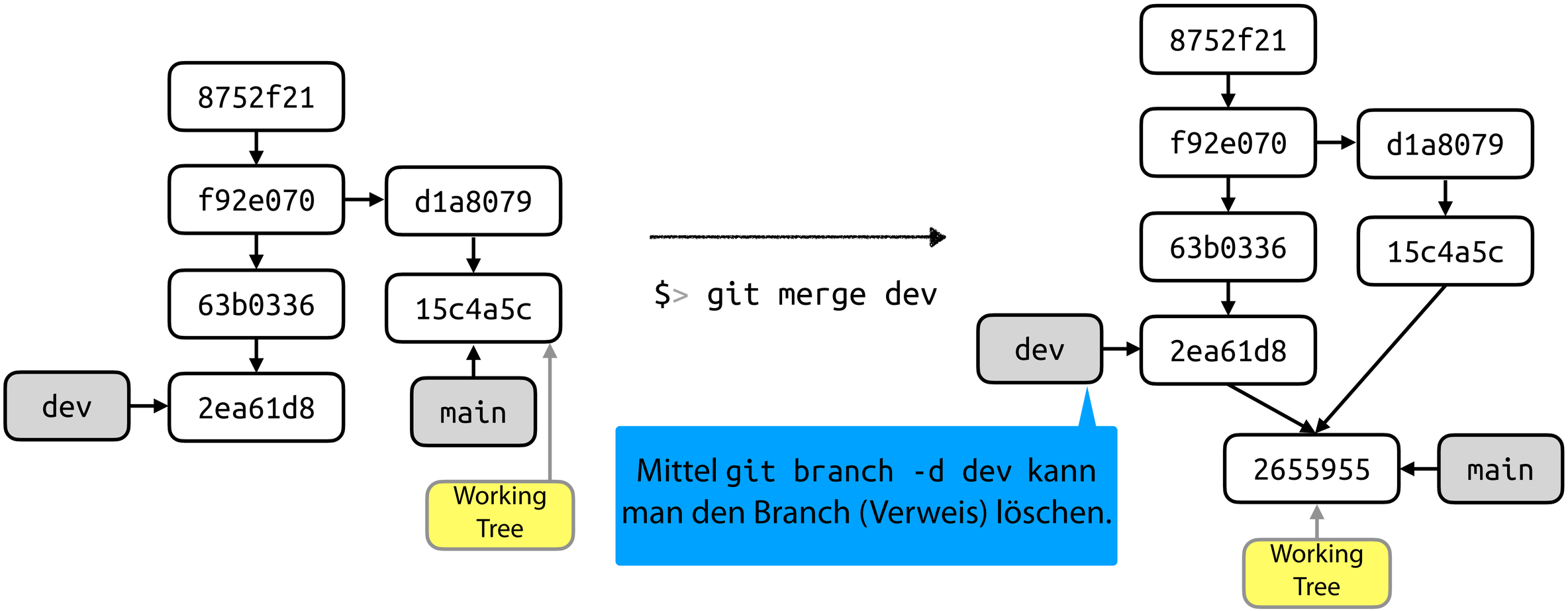
Branches: Zusammenführung



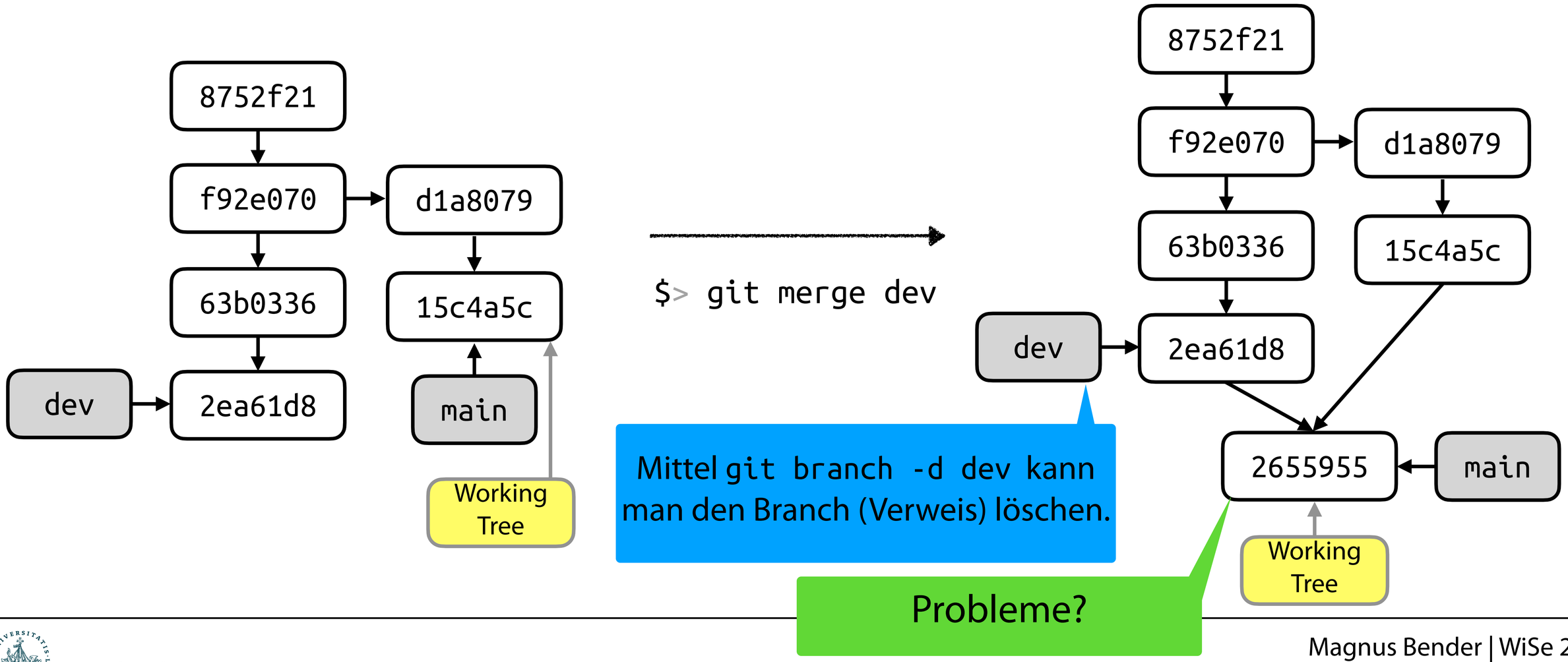
Branches: Zusammenführung



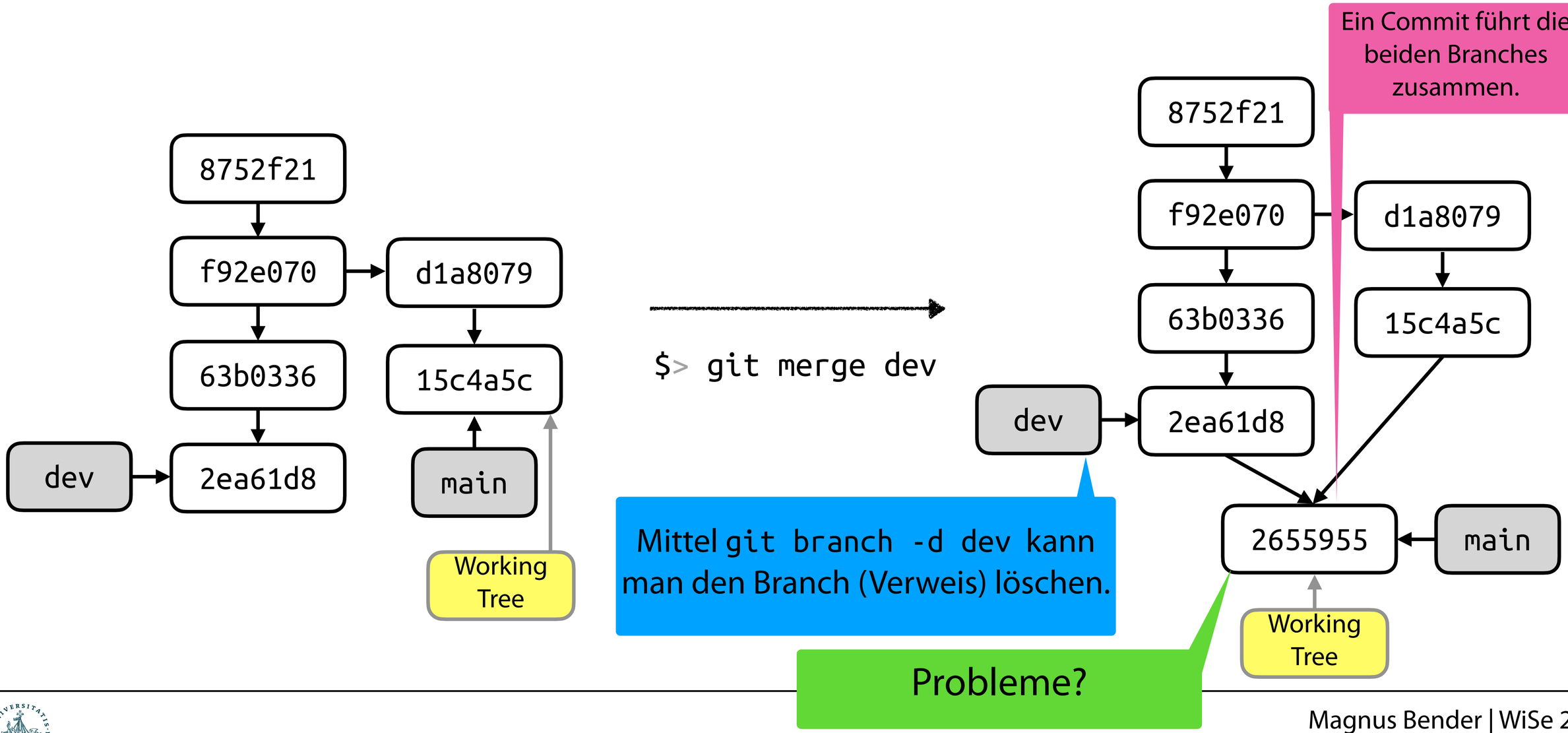
Branches: Zusammenführung



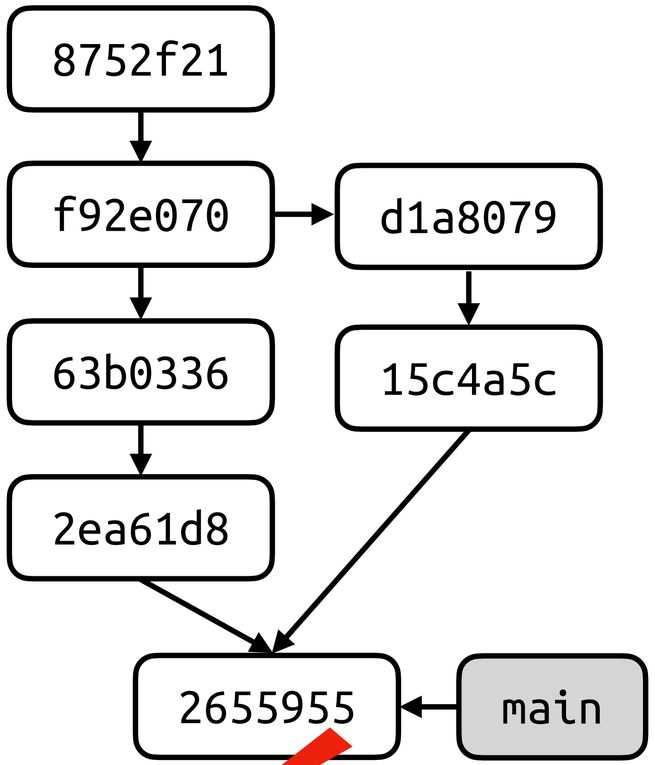
Branches: Zusammenführung



Branches: Zusammenführung

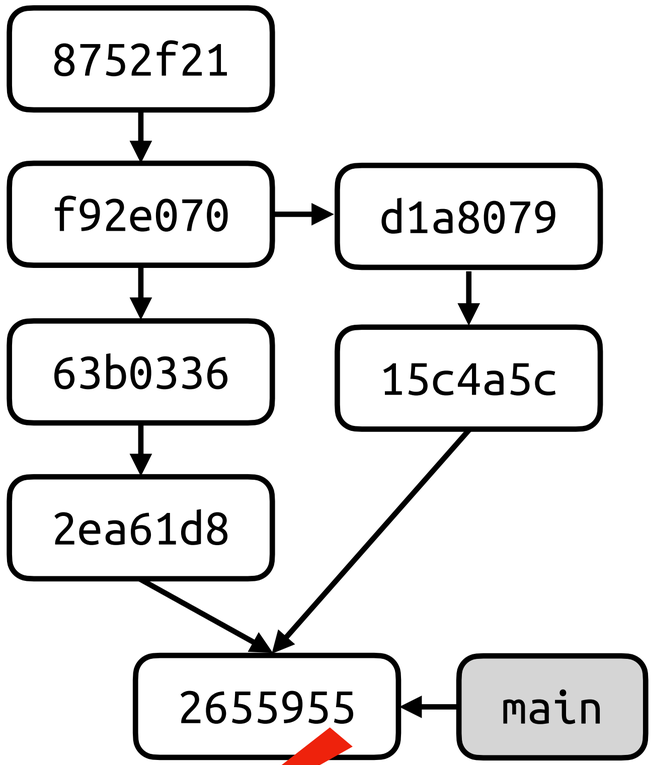


Zusammenführung: Konflikte



Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

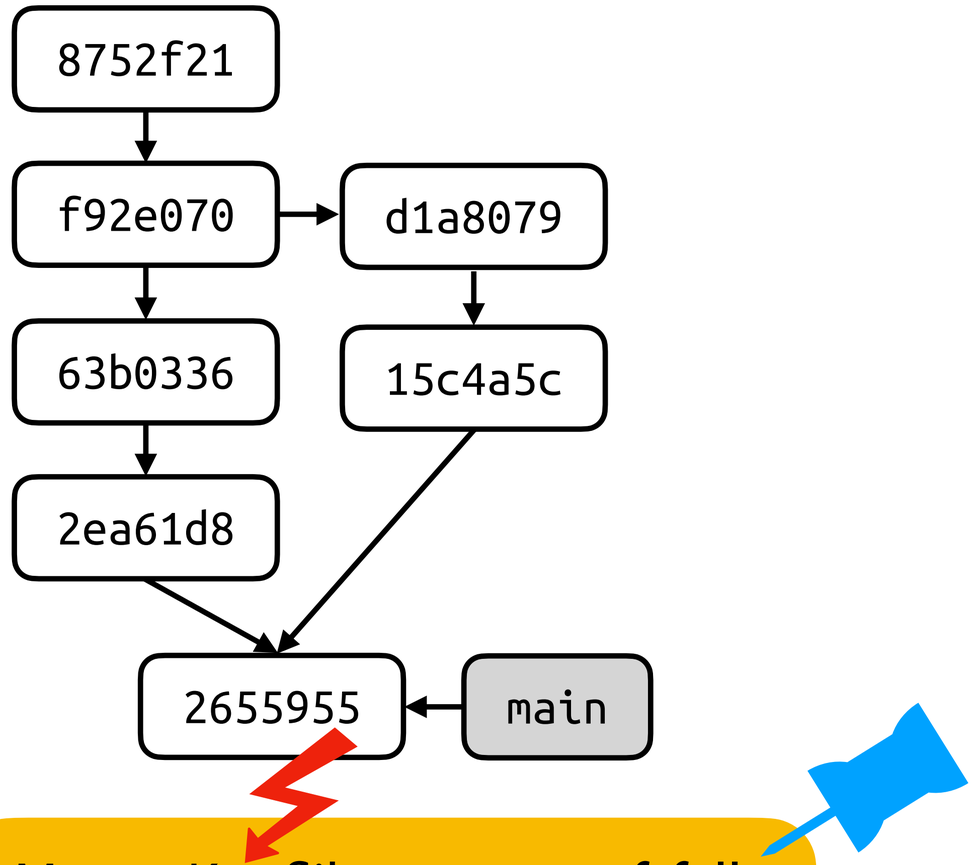
Zusammenführung: Konflikte




```
$> git merge dev
  # Fehlermeldung
```

Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

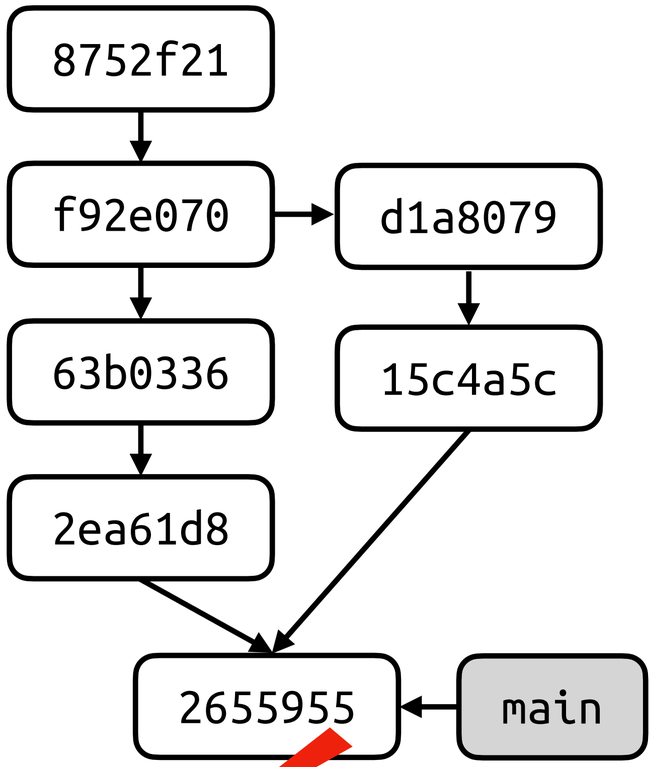
Zusammenführung: Konflikte




```
$> git merge dev
  # Fehlermeldung
$> git status
  # Problematische Dateien werden angezeigt
```

Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

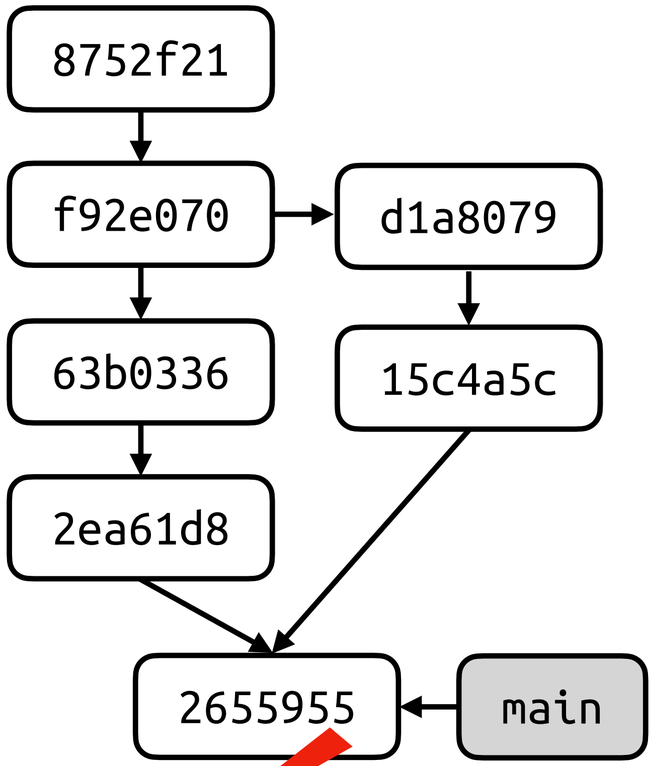
Zusammenführung: Konflikte




```
$> git merge dev
  # Fehlermeldung
$> git status
  # Problematische Dateien werden angezeigt

$> vim DateiMitFehler.txt
  # Konflikt in Datei beheben
```

Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

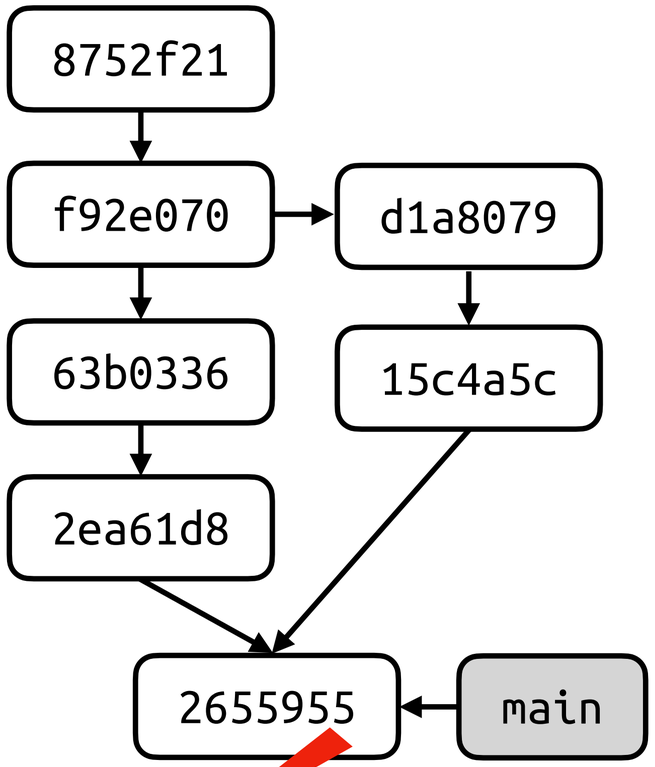
Zusammenführung: Konflikte




```
$> git merge dev
  # Fehlermeldung
$> git status
  # Problematische Dateien werden angezeigt

$> vim DateiMitFehler.txt
  # Konflikt in Datei beheben
$> …
```

Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

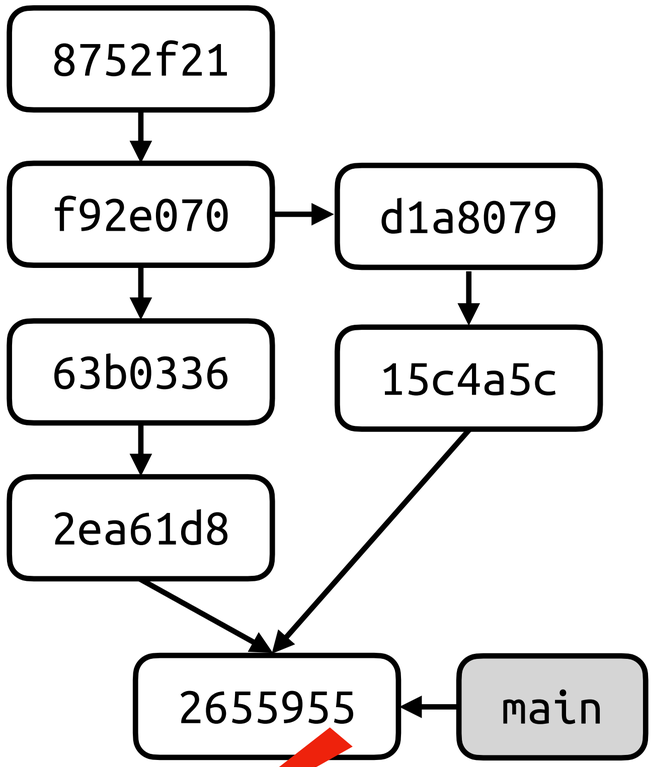
Zusammenführung: Konflikte




```
$> git merge dev
  # Fehlermeldung
$> git status
  # Problematische Dateien werden angezeigt

$> vim DateiMitFehler.txt
  # Konflikt in Datei beheben
$> …

$> git add DateiMitFehler.txt
```

Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

Zusammenführung: Konflikte

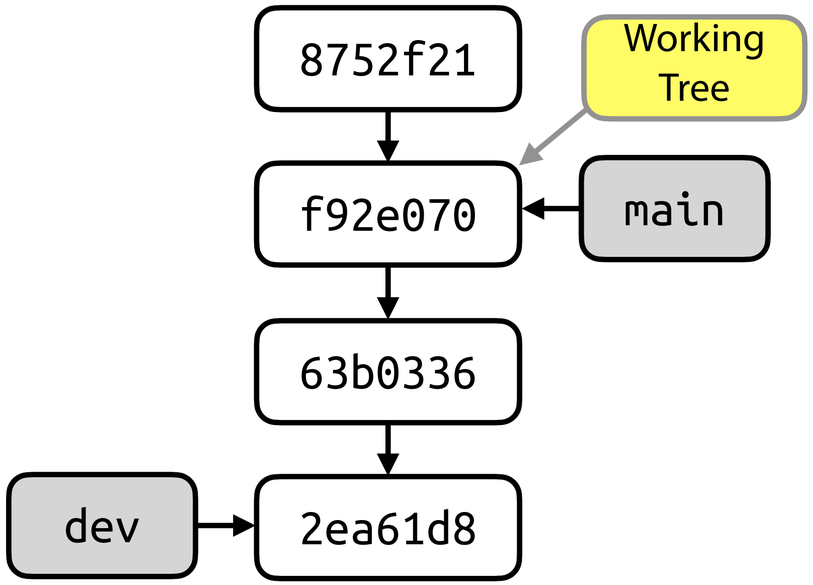



```
$> git merge dev
  # Fehlermeldung
$> git status
  # Problematische Dateien werden angezeigt

$> vim DateiMitFehler.txt
  # Konflikt in Datei beheben
$> …

$> git add DateiMitFehler.txt

$> git commit
```

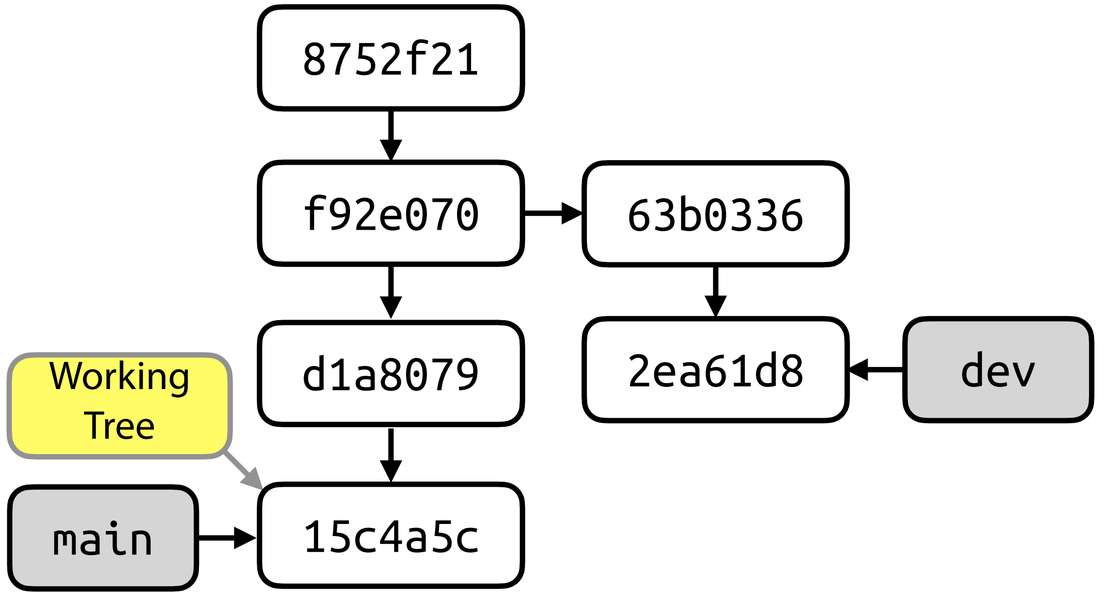
Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

Weitere Zusammenführungen

Fast-Forward

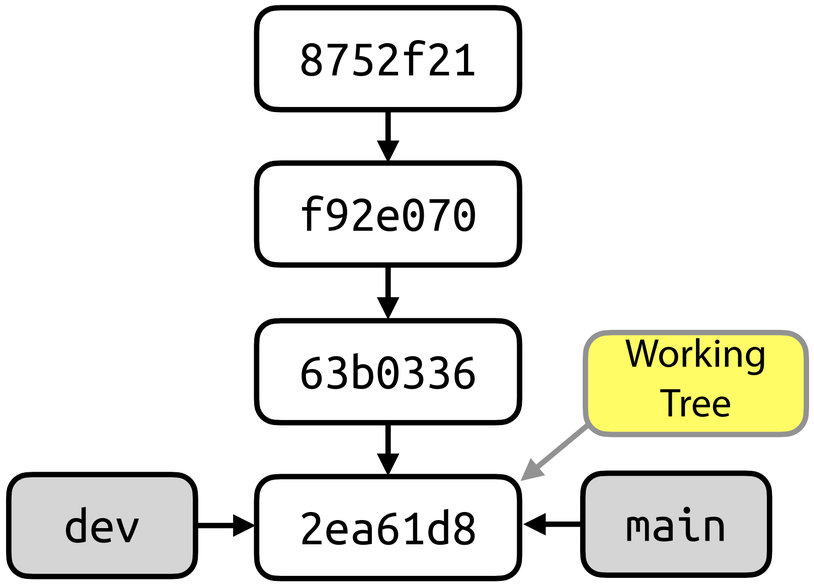


Rebase

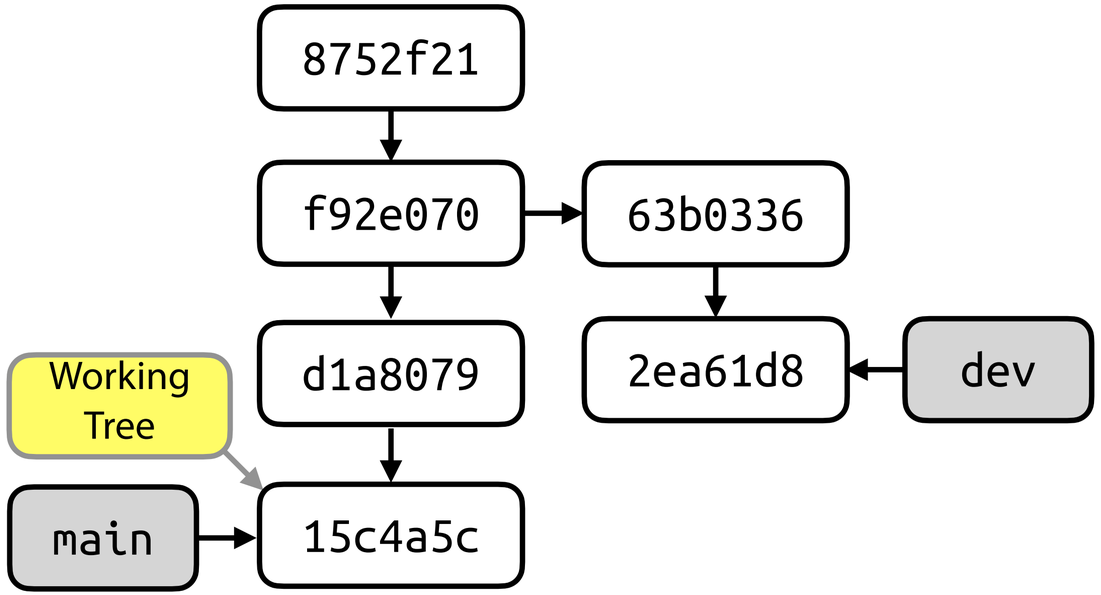


Weitere Zusammenführungen

Fast-Forward

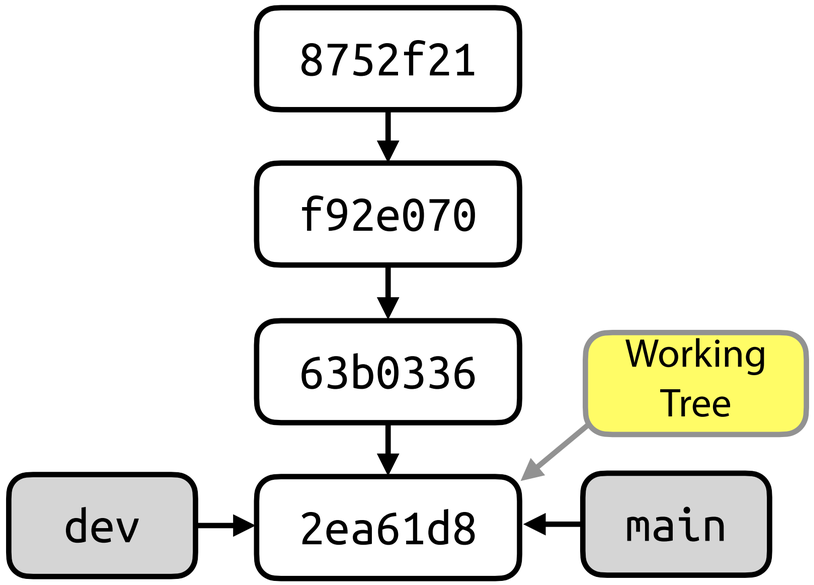


Rebase



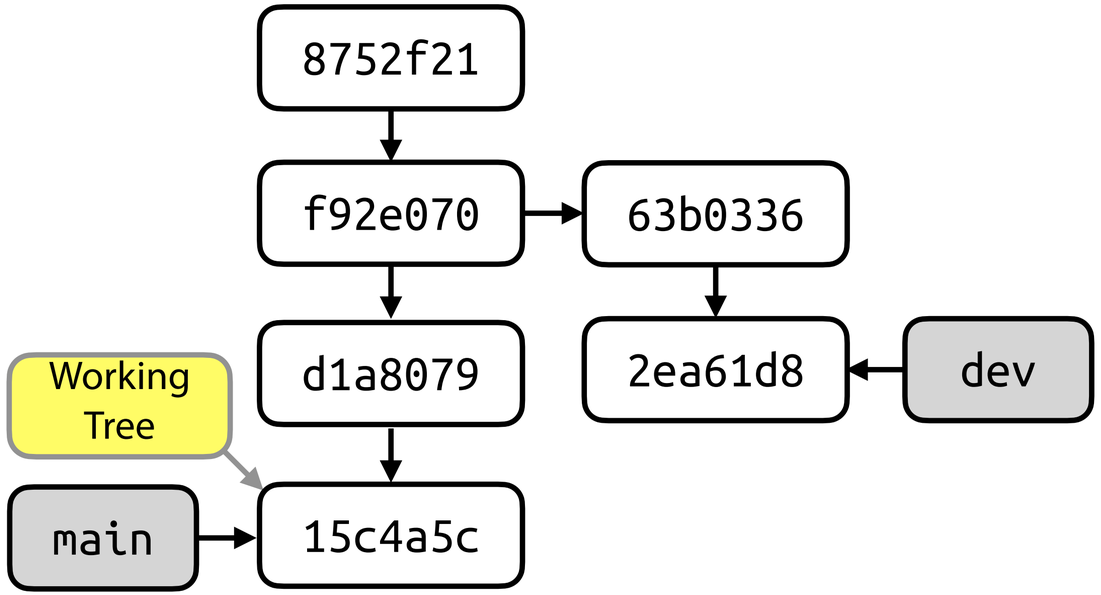
Weitere Zusammenführungen

Fast-Forward



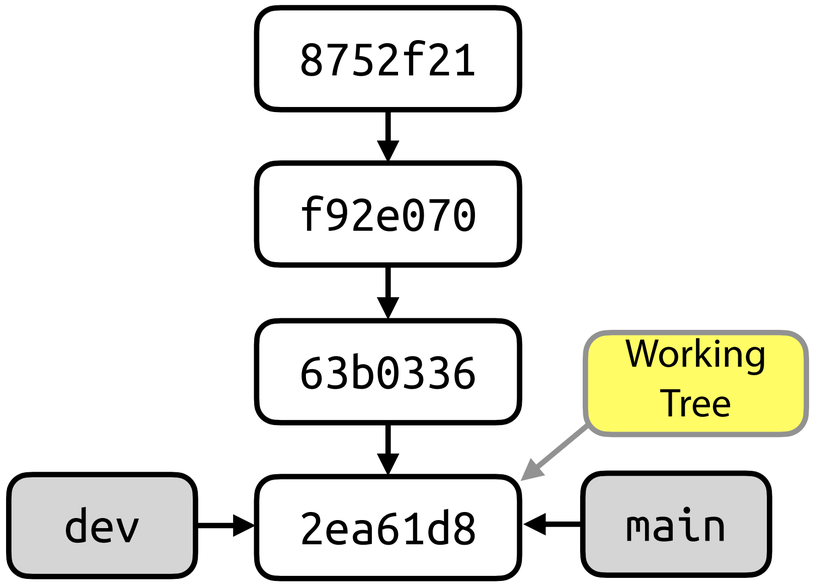

```
$> git merge dev
```

Rebase



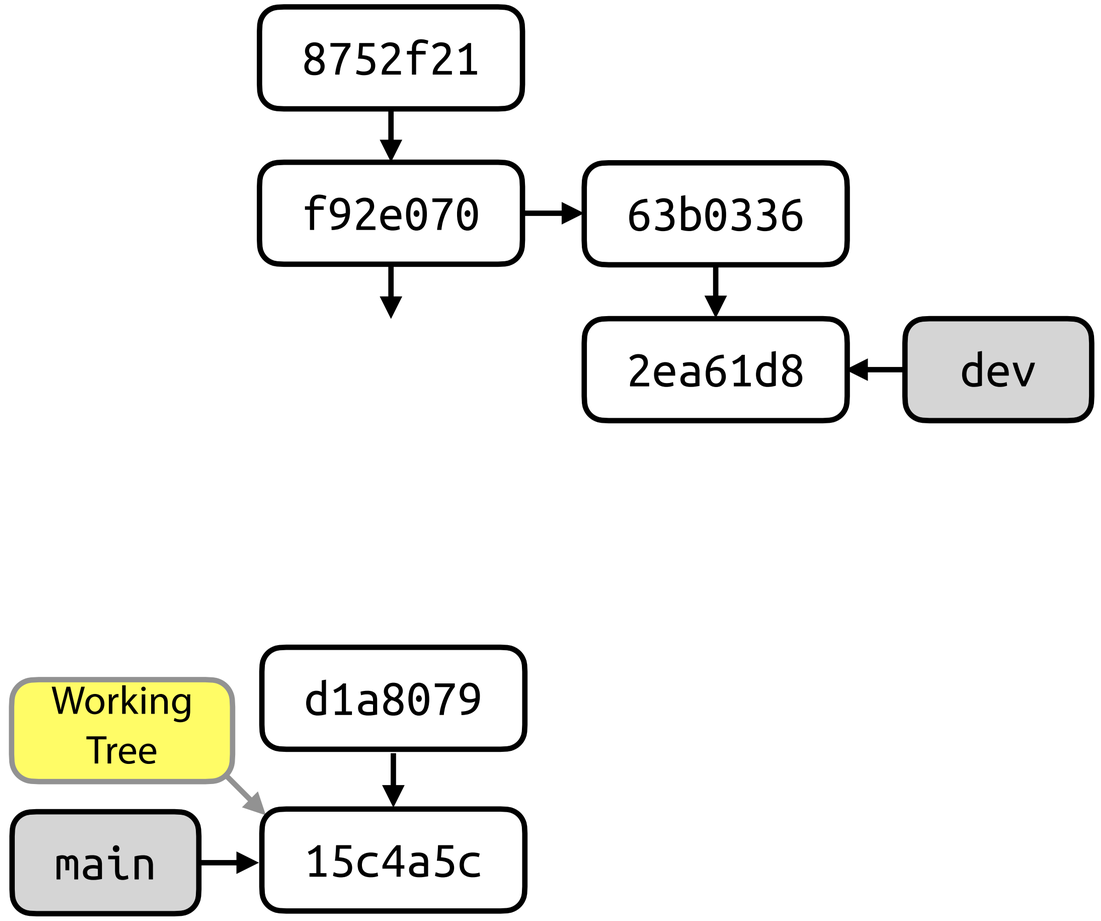
Weitere Zusammenführungen

Fast-Forward



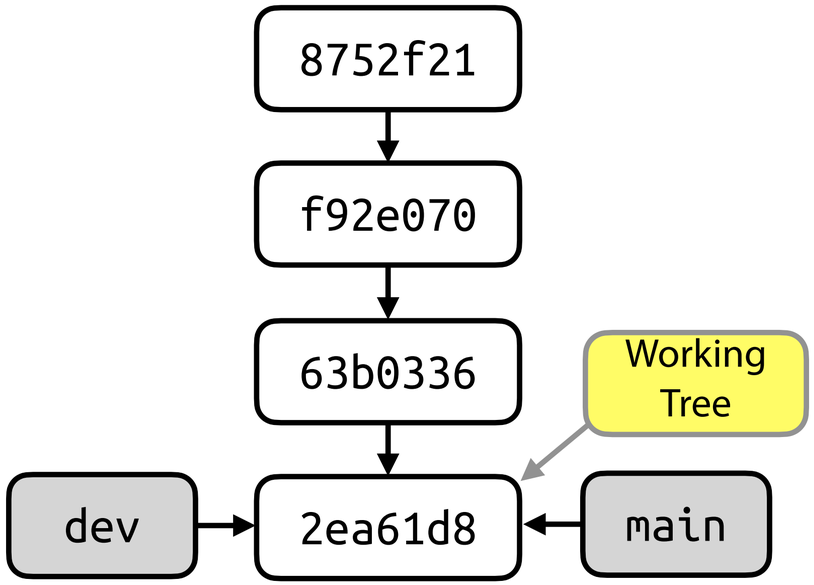

```
$> git merge dev
```

Rebase



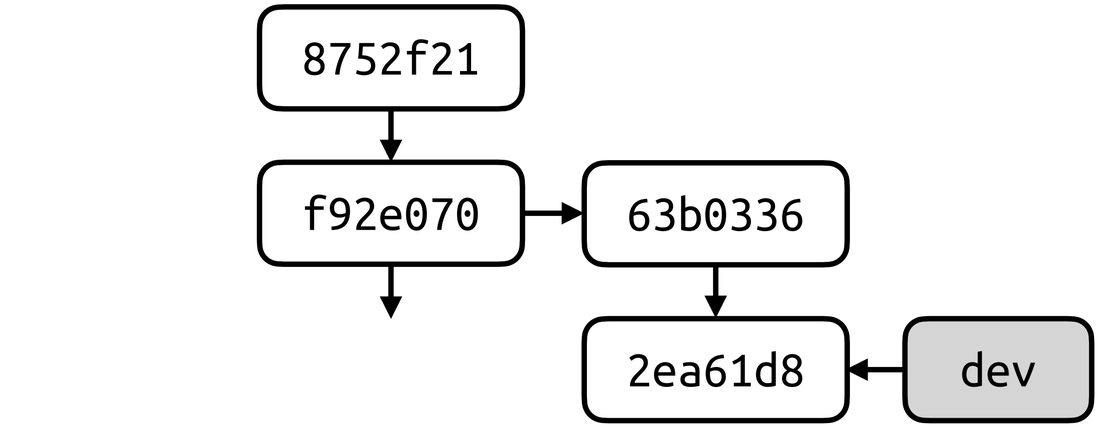
Weitere Zusammenführungen

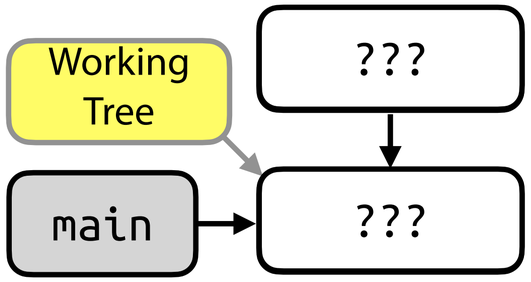
Fast-Forward




```
$> git merge dev
```

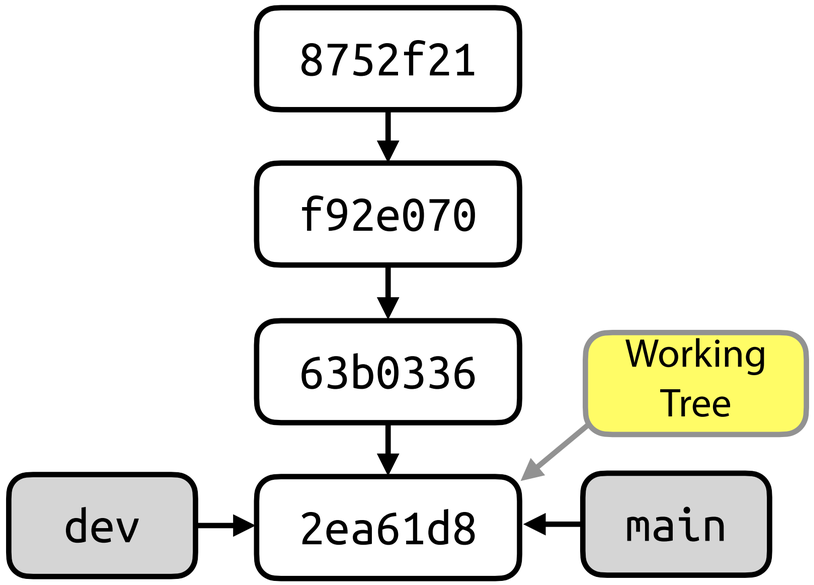
Rebase





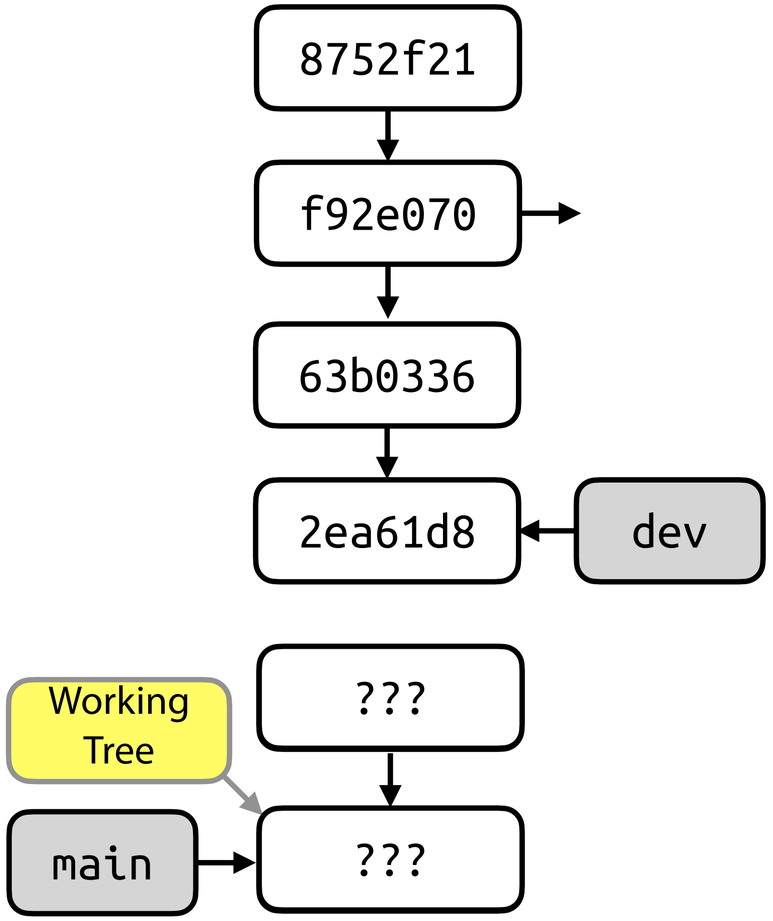
Weitere Zusammenführungen

Fast-Forward



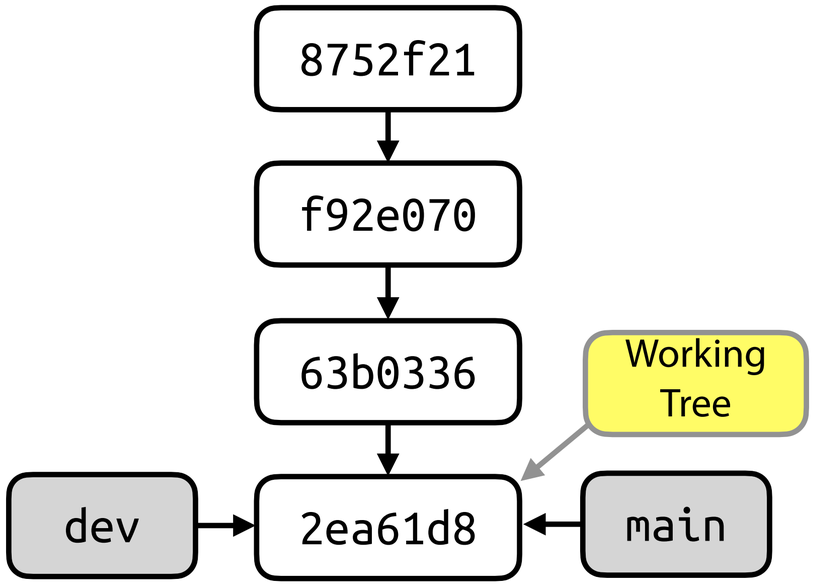

```
$> git merge dev
```

Rebase



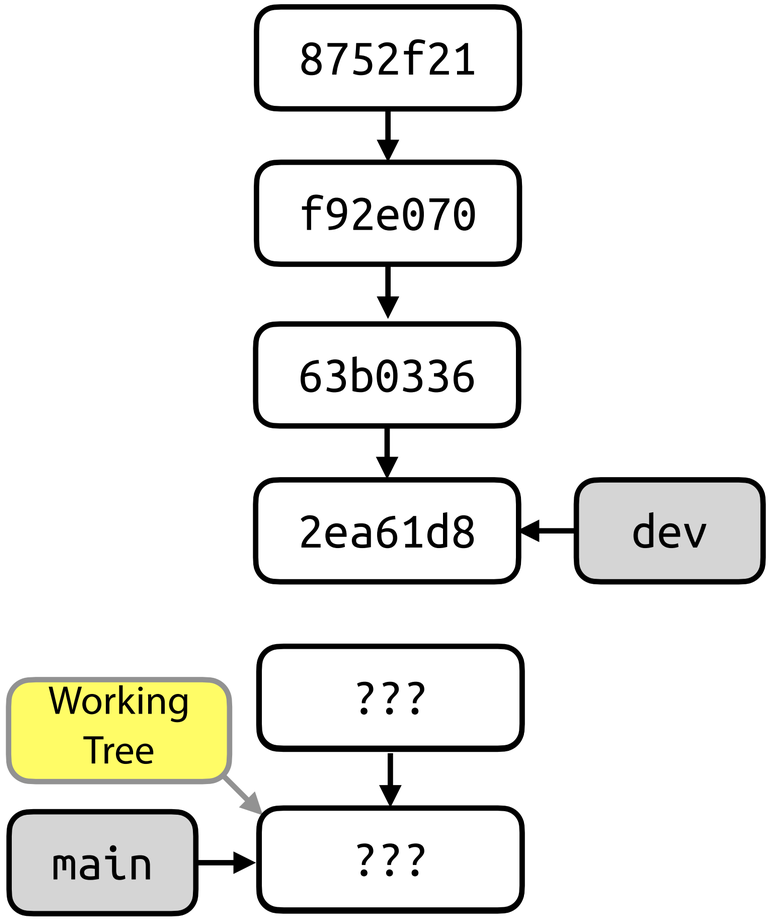
Weitere Zusammenführungen

Fast-Forward



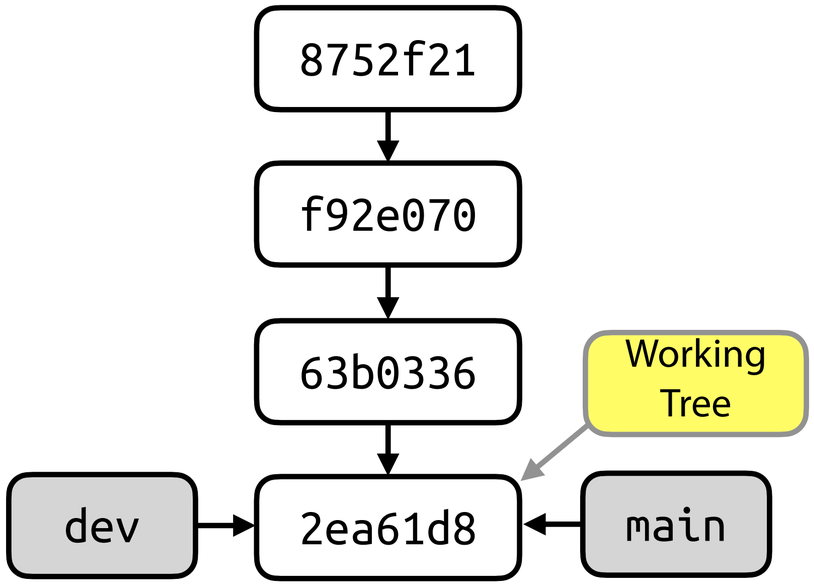

```
$> git merge dev
```

Rebase



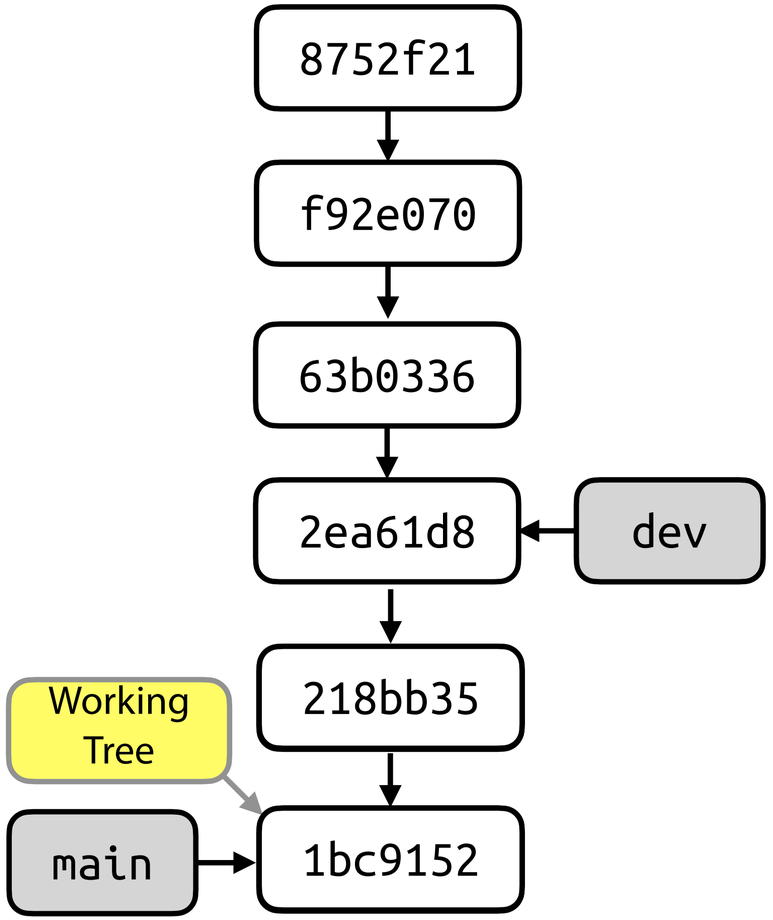
Weitere Zusammenführungen

Fast-Forward



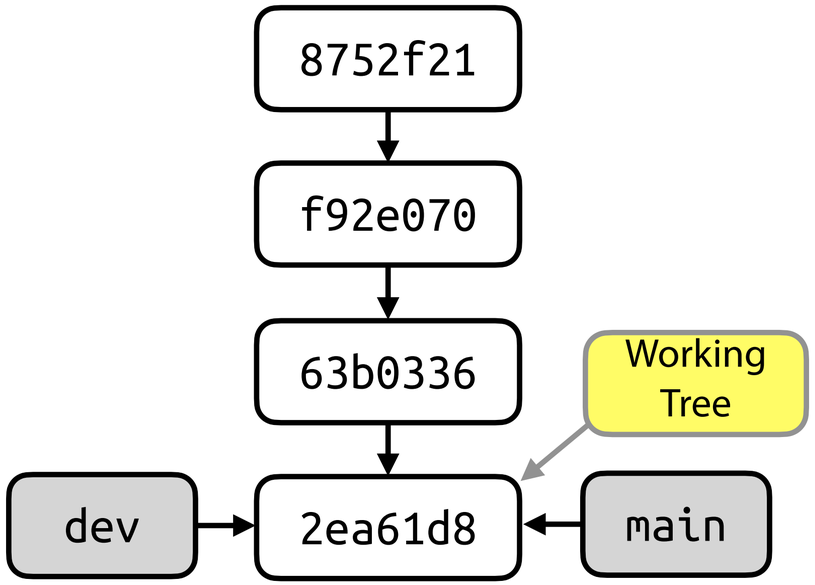

```
$> git merge dev
```

Rebase



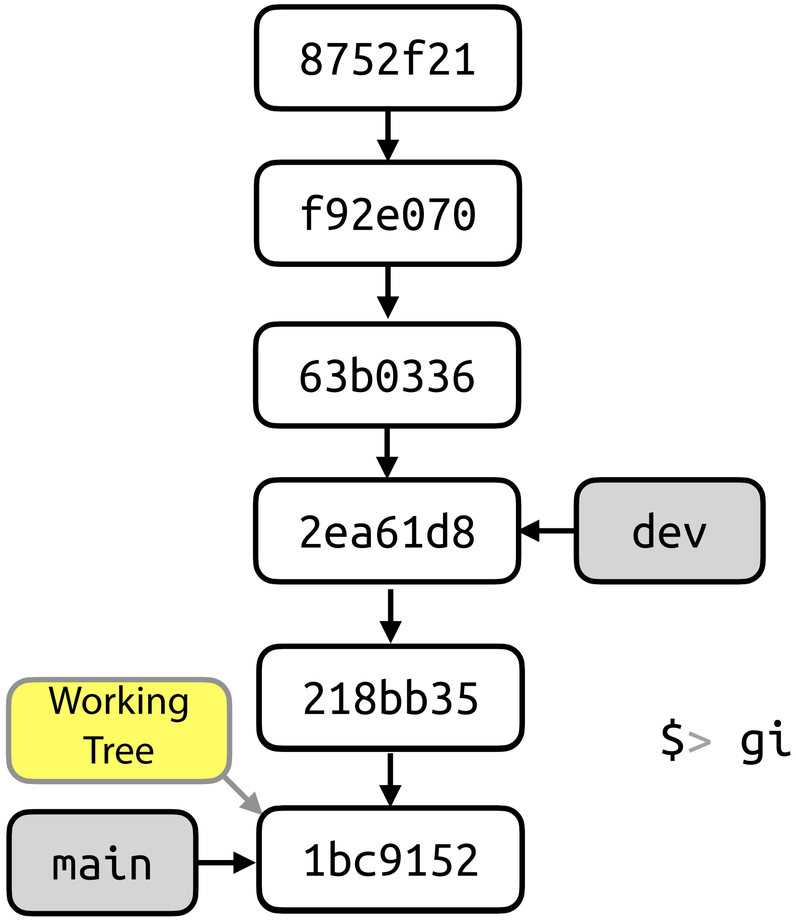
Weitere Zusammenführungen

Fast-Forward




```
$> git merge dev
```

Rebase




```
$> git rebase dev
```

„Gitignore“

- Binäre- und andere Nicht-Textdateien kann Git nur schlecht verwalten

- Kompilierte Programme, LaTeX-PDFs möchte man daher nicht im Repository haben

„Gitignore“

- Binäre- und andere Nicht-Textdateien kann Git nur schlecht verwalten
- Kompilierte Programme, LaTeX-PDFs möchte man daher nicht im Repository haben

.gitignore

```
.DS_Store
__pycache__

*.aux
*.fdb_latexmk
*.log
*.pdf

/data/*
/bin/*
```

„Gitignore“

- Binäre- und andere Nicht-Textdateien kann Git nur schlecht verwalten

Funktioniert ganz normal, aber Merge-Konflikte sind dann schwer zu lösen!

- Kompilierte Programme, LaTeX-PDFs möchte man daher nicht im Repository haben

.gitignore

```
.DS_Store
__pycache__

*.aux
*.fdb_latexmk
*.log
*.pdf

/data/*
/bin/*
```

„Gitignore“

- Binäre- und andere Nicht-Textdateien kann Git nur schlecht verwalten

Funktioniert ganz normal, aber Merge-Konflikte sind dann schwer zu lösen!

- Kompilierte Programme, LaTeX-PDFs möchte man daher nicht im Repository haben

.gitignore

```
.DS_Store
__pycache__

*.aux
*.fdb_latexmk
*.log
*.pdf

/data/*
/bin/*
```

Bestimmte Datei-/ Ordnernamen ausschließen

Auch hier Glob-Pattern unterstütz

Explizit einen Pfad ausschließen

Anforderungen Versionsverwaltung

- ☑ Verlauf der Änderungen (textbasierte Dateien)
- ☑ Verschiedene **Entwicklungszweige** gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte
- ☑ Zusammenführen von **Entwicklungszweigen**
- ☑ Ein (zentrales) Repository

Anforderungen Versionsverwaltung

- ☑ Verlauf der Änderungen (textbasierte Dateien)
- ☑ Verschiedene **Entwicklungszweige** gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte

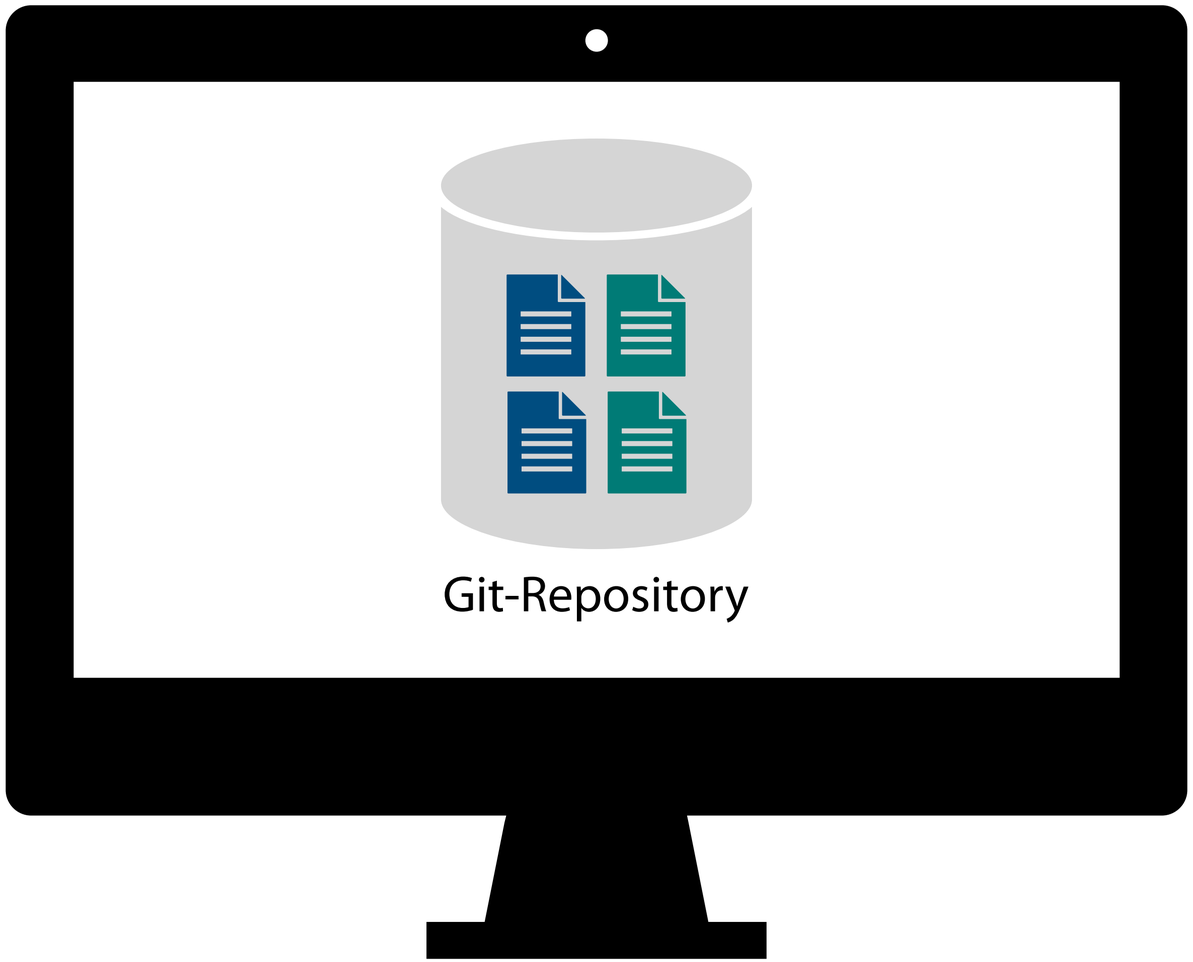
Naja, wir arbeiten immer im lokalen Repository!

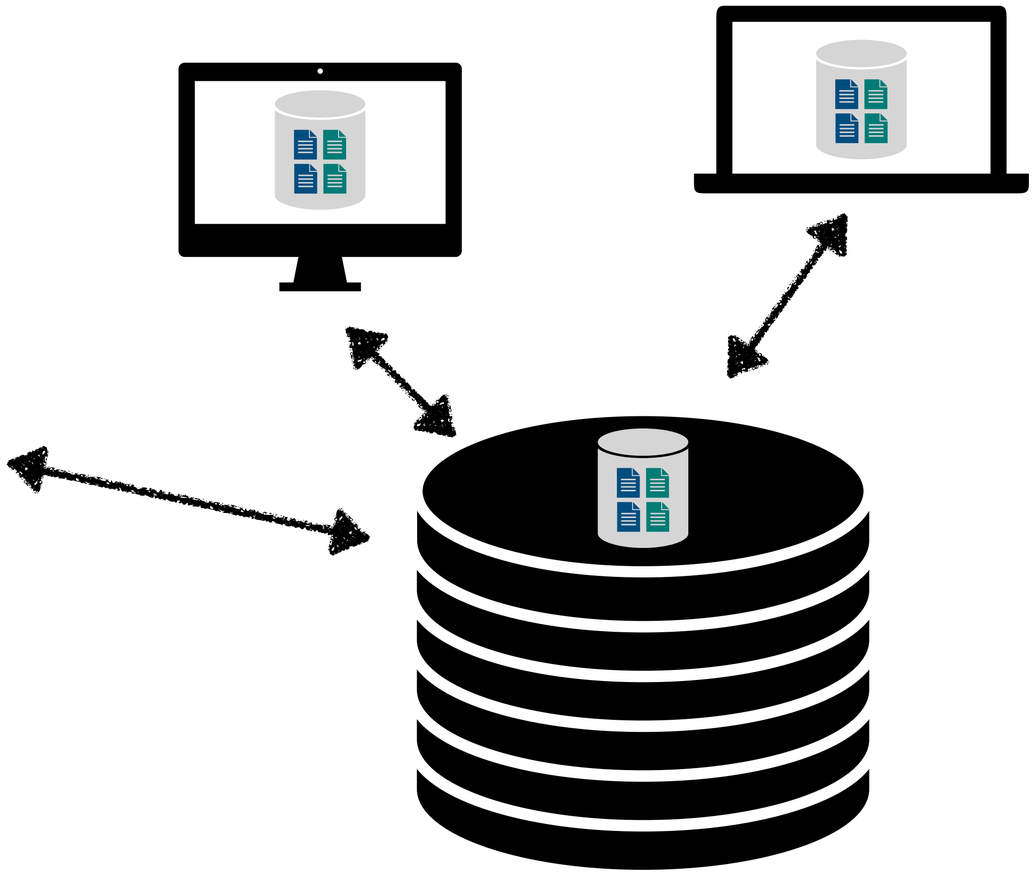
- ☑ Zusammenführen von **Entwicklungszweigen**
- ☑ Ein (zentrales) Repository

II.
Git

3. Remote: Push, Pull

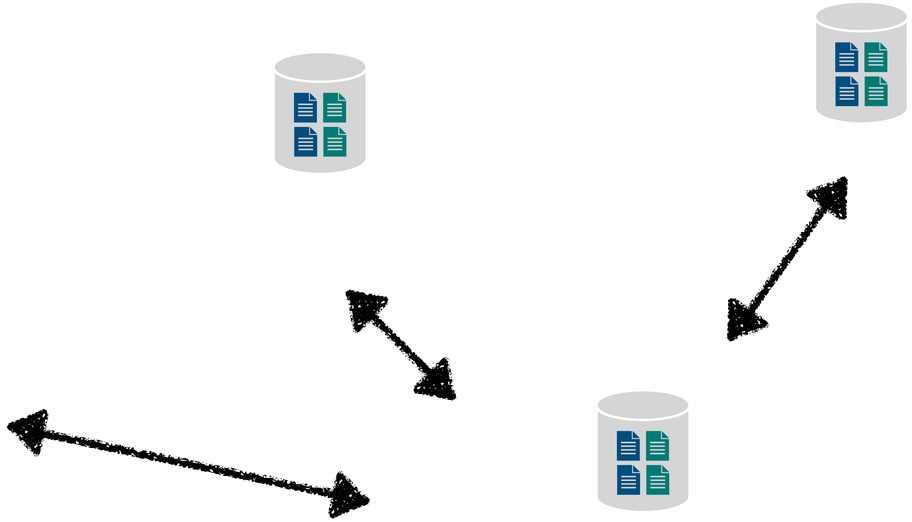
Verteilte Repositories





Verteilte Repositories

Git-Repository



Remote Repository

- Zugriff über eine URL
  - Zugriff über Netzwerk oder auch lokal möglich
  - Lese- und/ oder Schreibbrechte

Remote Repository

- Zugriff über eine URL
  - Zugriff über Netzwerk oder auch lokal möglich
  - Lese- und/ oder Schreibbrechte
- Verschiedene Protokolle
  - SSH — Erfordert eine Authentifikation
  - HTTP(S) — Erfordert eine Authentifikation nur beim „Hochladen“

Remote Repository

- Zugriff über eine URL
  - Zugriff über Netzwerk oder auch lokal möglich
  - Lese- und/ oder Schreibbrechte
- Verschiedene Protokolle
  - SSH — Erfordert eine Authentifikation
  - HTTP(S) — Erfordert eine Authentifikation nur beim „Hochladen“

```
$> git remote add NAME URL

$> git remote add origin https://github.com/torvalds/linux.git

$> git remote add MyCopy git@github.com:myuser/linux.git
```

Remote Repository

- Zugriff über eine URL
  - Zugriff über Netzwerk oder auch lokal möglich
  - Lese- und/ oder Schreibbrechte
- Verschiedene Protokolle
  - SSH — Erfordert eine Authentifikation
  - HTTP(S) — Erfordert eine Authentifikation nur beim „Hochladen“

```
$> git remote add NAME URL
$> git remote add origin https://github.com/torvalds/linux.git
$> git remote add MyCopy git@github.com:myuser/linux.git
```

Server: `https://github.com`
Nutzer/ Besitzer: `/torvalds`
Repository: `/linux.git`

Remote Repository

- Zugriff über eine URL
  - Zugriff über Netzwerk oder auch lokal möglich
  - Lese- und/ oder Schreibbrechte
- Verschiedene Protokolle
  - SSH — Erfordert eine Authentifikation
  - HTTP(S) — Erfordert eine Authentifikation nur beim „Hochladen“

```
$> git remote add NAME URL
$> git remote add origin https://github.com/torvalds/linux.git
$> git remote add MyCopy git@github.com:myuser/linux.git
```

Server, Nutzer/Besitzer, Repository

Server, Nutzer/Besitzer, Repository

Remote Repository

- Zugriff über eine URL
  - Zugriff über Netzwerk oder auch lokal möglich
  - Lese- und/ oder Schreibbrechte
- Verschiedene Protokolle
  - SSH — Erfordert eine Authentifikation
  - HTTP(S) — Erfordert eine Authentifikation nur beim „Hochladen“

```
$> git remote add NAME URL
$> git remote add origin https://github.com/torvalds/linux.git
$> git remote add MyCopy git@github.com:myuser/linux.git
```

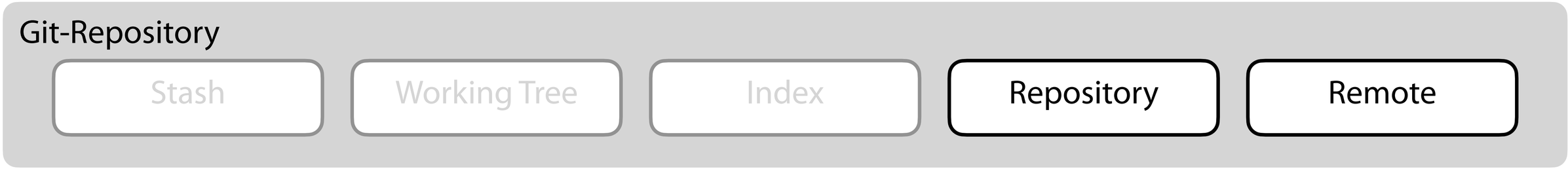
Server, Nutzer/Besitzer, Repository

Quelle, üblicherweise benannt als `origin`
Schreiben i.A. nicht erlaubt

Server, Nutzer/Besitzer, Repository

Eigene Kopie, benannt als `MyCopy`
Schreiben i.A. möglich

Herunterladen



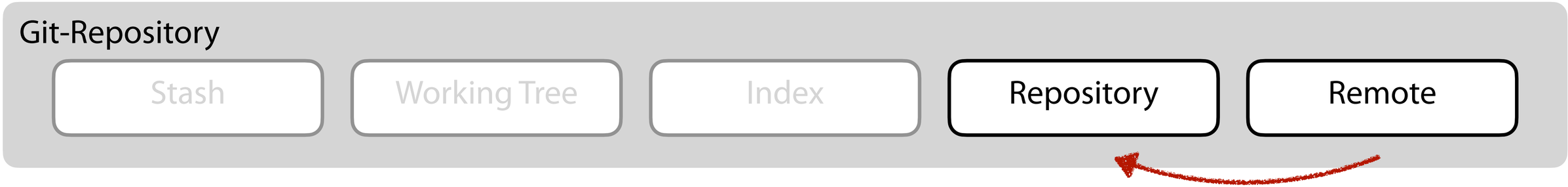
Herunterladen

Herunterladen


```
$> git fetch --all
  remote: Enumerating objects: 513, done.
  remote: Counting objects: 100% (116/116), done.
  remote: Compressing objects: 100% (107/107), done.
  remote: Total 513 (delta 48), reused 0 (delta 0), pack-reused 397
  Receiving objects: 100% (513/513), 271.85 KiB | 1.93 MiB/s, done.
  Resolving deltas: 100% (289/289), done.
  From https://server/user/repo
  * [new branch]      main     -> origin/main
$> git merge origin/main
```

Es können natürlich Merge-Konflikte auftreten.

Herunterladen

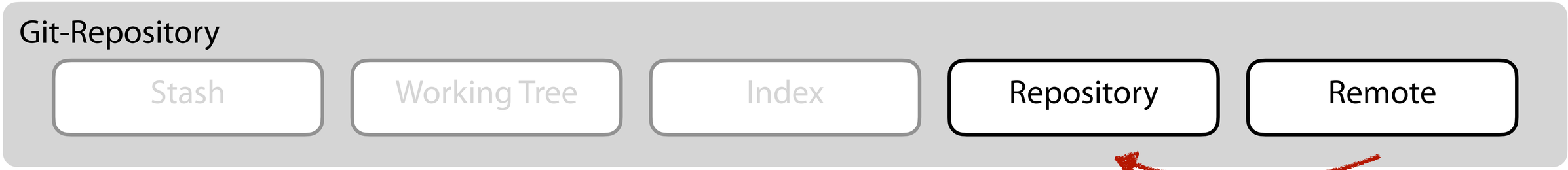


`git fetch [--all | *REMOTE BRANCH*]`
`git pull *REMOTE BRANCH*`

```
$> git fetch --all
  remote: Enumerating objects: 513, done.
  remote: Counting objects: 100% (116/116), done.
  remote: Compressing objects: 100% (107/107), done.
  remote: Total 513 (delta 48), reused 0 (delta 0), pack-reused 397
  Receiving objects: 100% (513/513), 271.85 KiB | 1.93 MiB/s, done.
  Resolving deltas: 100% (289/289), done.
  From https://server/user/repo
  * [new branch]      main     -> origin/main
$> git merge origin/main
```

Es können natürlich Merge-Konflikte auftreten.

Herunterladen



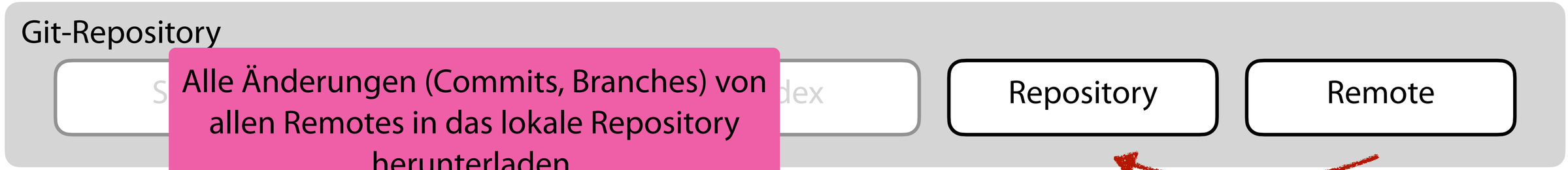
`git fetch [--all | *REMOTE BRANCH*]`
`git pull *REMOTE BRANCH*`

```
$> git fetch --all
  remote: Enumerating objects: 513, done.
  remote: Counting objects: 100% (116/116), done.
  remote: Compressing objects: 100% (107/107), done.
  remote: Total 513 (delta 48), reused 0 (delta 0), pack-reused 397
  Receiving objects: 100% (513/513), 271.85 KiB | 1.93 MiB/s, done.
  Resolving deltas: 100% (289/289), done.
  From https://server/user/repo
  * [new branch]      main     -> origin/main
$> git merge origin/main
```

`$> git pull origin main`

Herunterladen

Es können natürlich Merge-Konflikte auftreten.



Alle Änderungen (Commits, Branches) von allen Remotes in das lokale Repository herunterladen.
(Keine Änderung am Working Tree)

```
git fetch [--all | REMOTE BRANCH]
git pull REMOTE BRANCH
```

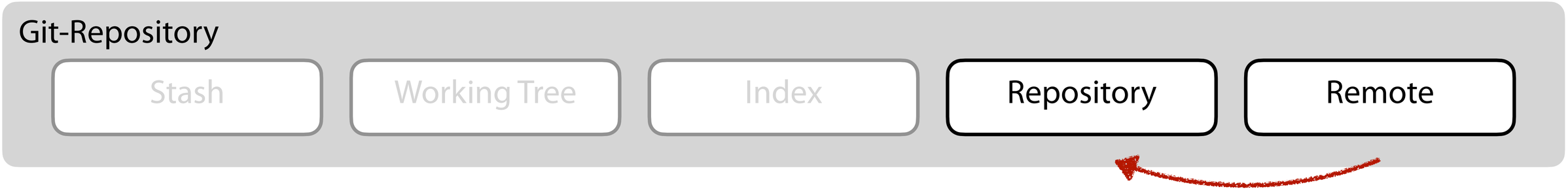
```
$> git fetch --all
  remote: Enumerating objects: 513, done.
  remote: Counting objects: 100% (116/116), done.
  remote: Compressing objects: 100% (107/107), done.
  remote: Total 513 (delta 48), reused 0 (delta 0), pack-reused 397
  Receiving objects: 100% (513/513), 271.85 KiB | 1.93 MiB/s, done.
  Resolving deltas: 100% (289/289), done.
  From https://server/user/repo
  * [new branch]      main       -> origin/main
$> git merge origin/main
```

```
$> git pull origin main
```

Fetch und Merge in einem Schritt

Den aktuellen lokalen Branch mit einem remote Branch mergen.

Neu herunterladen

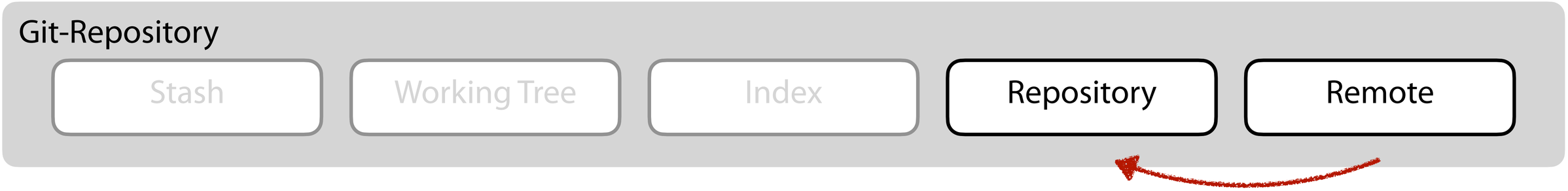


`git fetch [--all | *REMOTE BRANCH*]`
`git pull *REMOTE BRANCH*`

`git clone *URL*`

```
$> mkdir linux
$> cd ./linux/
$> git init
$> git remote add origin https://github.com/torvalds/linux.git
$> git pull origin master
```

Neu herunterladen

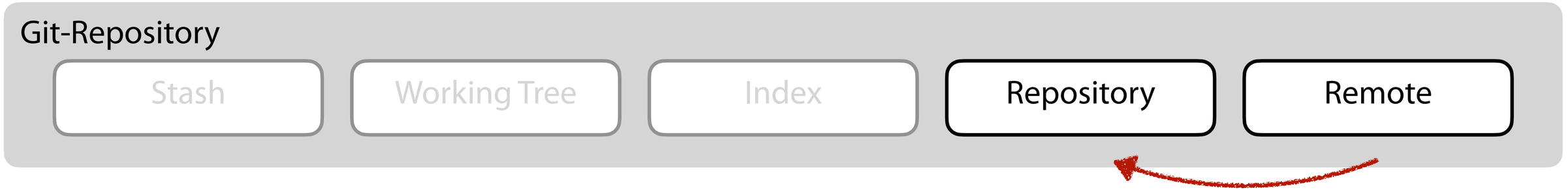



```
git fetch [--all | REMOTE BRANCH]
git pull REMOTE BRANCH

git clone URL
```

```
$> mkdir linux
$> cd ./linux/
$> git init
$> git remote add origin https://github.com/torvalds/linux.git
$> git pull origin master
```

```
$> git clone
https://github.com/
torvalds/linux.git
```

Hochladen




```
git fetch [--all | REMOTE BRANCH]
git pull REMOTE BRANCH

git clone URL
```

Hochladen


git push *REMOTE BRANCH*
Git-Repository
Stash
Working Tree
Index
Repository
Remote
git fetch [--all | *REMOTE BRANCH*]
git pull *REMOTE BRANCH*
git clone *URL*

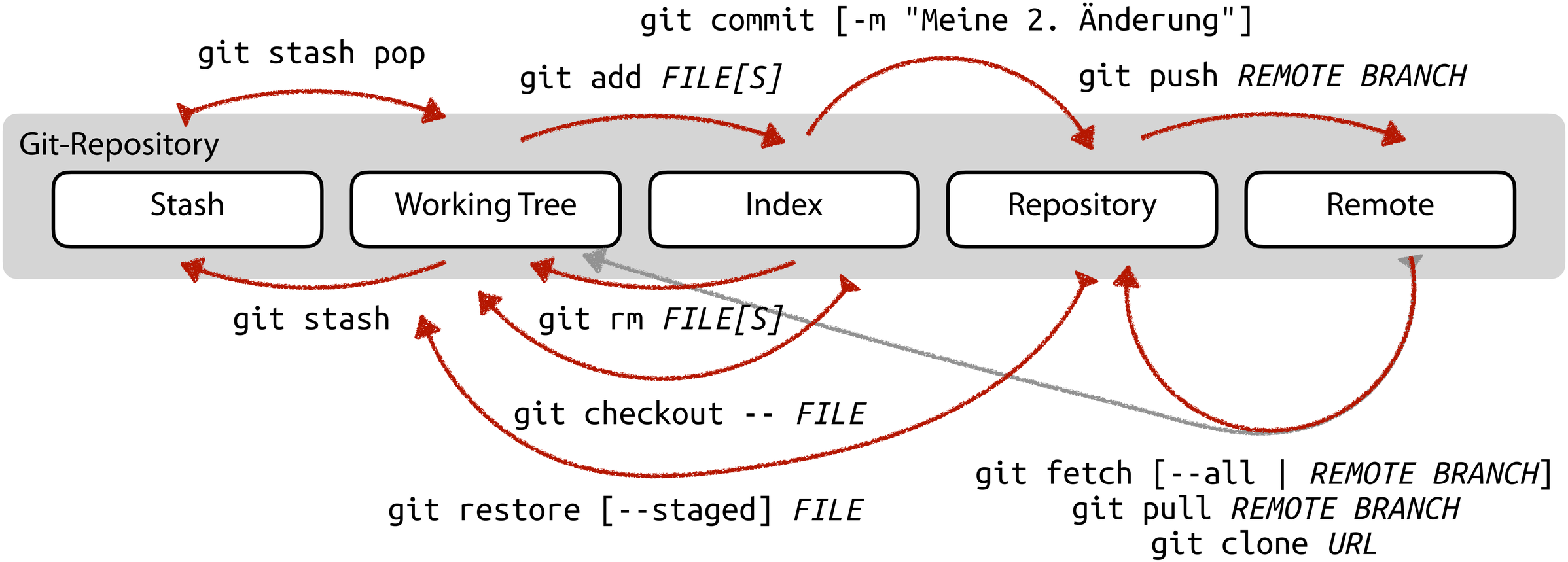
Hochladen


```
$> git commit -m "Meine Änderung"
$> git push MyCopy main
  Enumerating objects: 513, done.
  Counting objects: 100% (513/513), done.
  Delta compression using up to 12 threads
  Compressing objects: 100% (200/200), done.
  Writing objects: 100% (513/513), 271.84 KiB | 2.75 MiB/s, done.
  Total 513 (delta 289), reused 513 (delta 289), pack-reused 0
  remote: Resolving deltas: 100% (289/289), done.
  To server:user/repo.git
  * [new branch]      main -> main
```

Git Befehle I

Konfiguration	`git config [--global] user.name "NAME"`	Angabe des Namens, mit dem Commits unterschrieben werden
	`git config [--global] user.email "E-MAIL"`	Angabe der Mail-Adresse, die in Commits angegeben wird
Repository, Verlauf	`git init`	Erzeugen eines leere Repository
	`git log --oneline --graph`	Anzeige des Verlaufs der Commits
	`git checkout HASH`	Einen Commit im Working Tree öffnen
	`git checkout HEAD`	Zurück zum Kopf des Repositories
Working Tree, Index, Commit	`git add FILE[S]`	Eine Datei/ Pfad zum Commit vormerken (*staged*)
	`git status`	Status des Repository mit Dateienstatus anzeigen
	`git commit [-m "MESSAGE"]`	Erstellen eins Commit
	`git rm FILE[S]`	Löschen von Dateien aus dem Index (nicht auch alten Commits)
	`git checkout -- FILE`	Zurücksetzen einer Datei im Working Tree auf *staged* Version
	`git restore [--staged] FILE`	Zurücksetzen einer Datei im Working Tree auf letzten Commit
Stash	`git stash`	Working Tree im Zwischenspeicher ablegen
	`git stash pop`	Zwischenspeicher auf Working Tree anwenden
Branches, Merge	`git branch NAME`	Einen neuen Branch „hier“ erzeugen
	`git checkout BRANCHNAME`	Eine Branch im Working Tree öffnen
	`git merge BRANCHNAME`	Einen anderen Branch in den aktuellen Branch mergen
Remote	`git remote add NAME URL`	Eine Remote-Repository anbinden
	`git push REMOTE BRANCH`	Einen Branch in das Remote-Repository „hochladen“
	`git fetch [--all | REMOTE BRANCH]`	Einen Branch oder alles mit dem Remote-Repository „abgleichen“
	`git pull REMOTE BRANCH`	Eine Branch vom Remote-Repository in den Working Tree „herunterladen“
	`git clone URL`	Eine Repository von einer URL „herunterladen“ und lokal erstellen

Git Befehle II



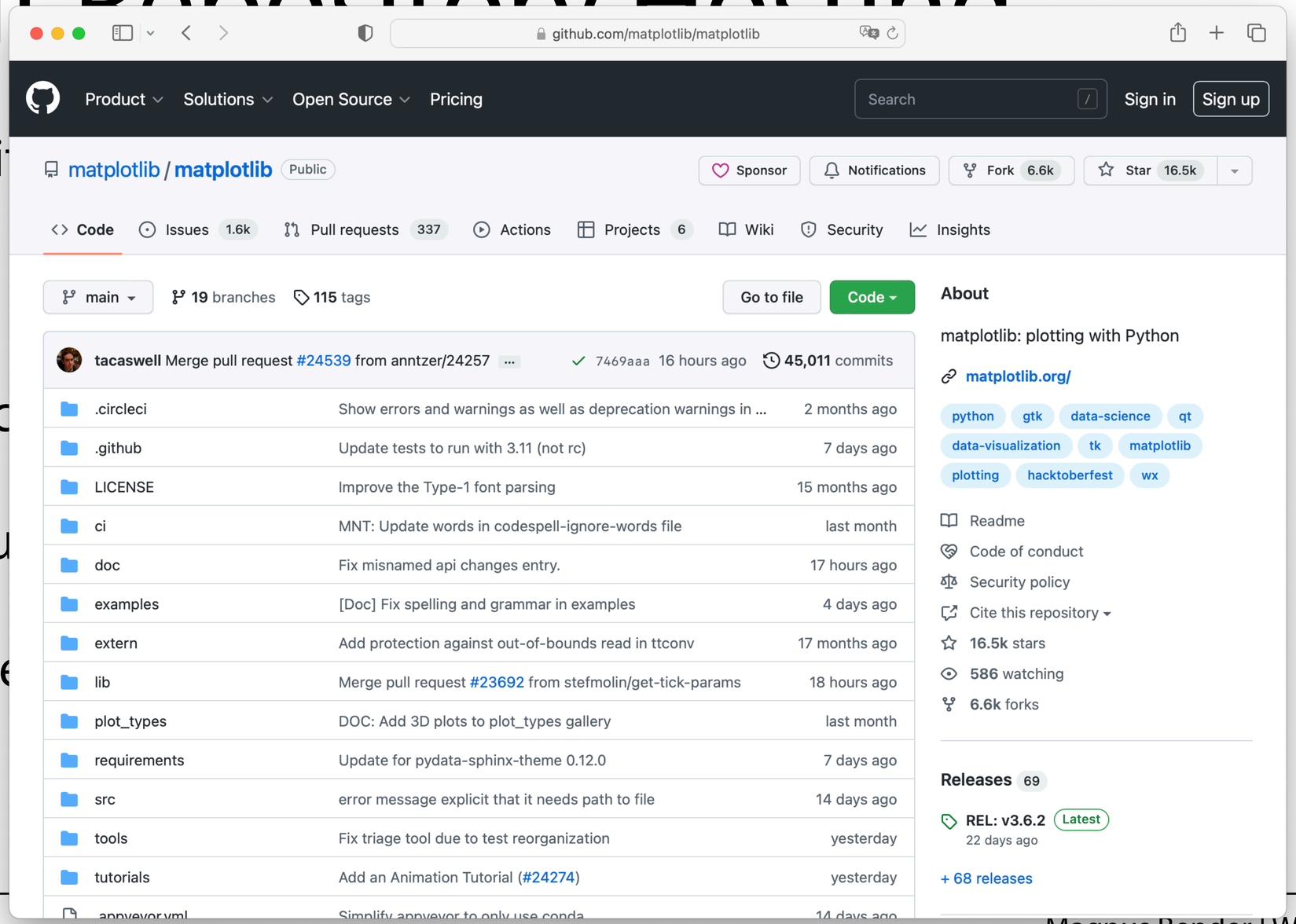
III.
GitHub

Git Repository Hosting

- GitHub, GitLab, Bitbucket, …
- Webinterface zur Darstellung des Codes
  - Commits, Branches, …
  - Forks, Pull Requests, …
- Projektmanagement
  - Issues, …

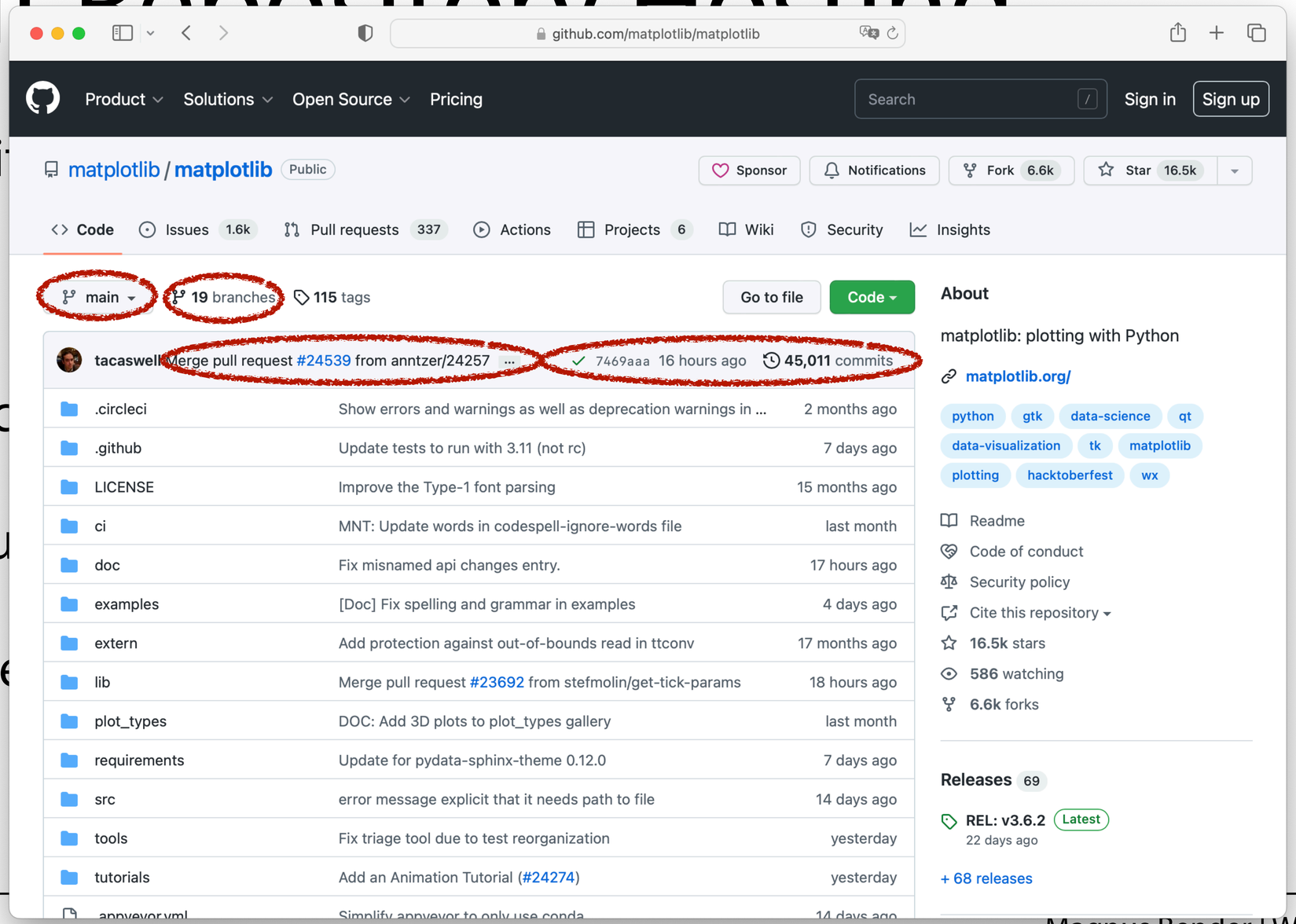
Git Repository Hosting

- GitHub, GitLab, Bi
- Webinterface zur
  - Commits, Branc
  - Forks, Pull Requ
- Projektmanageme
  - Issues, …



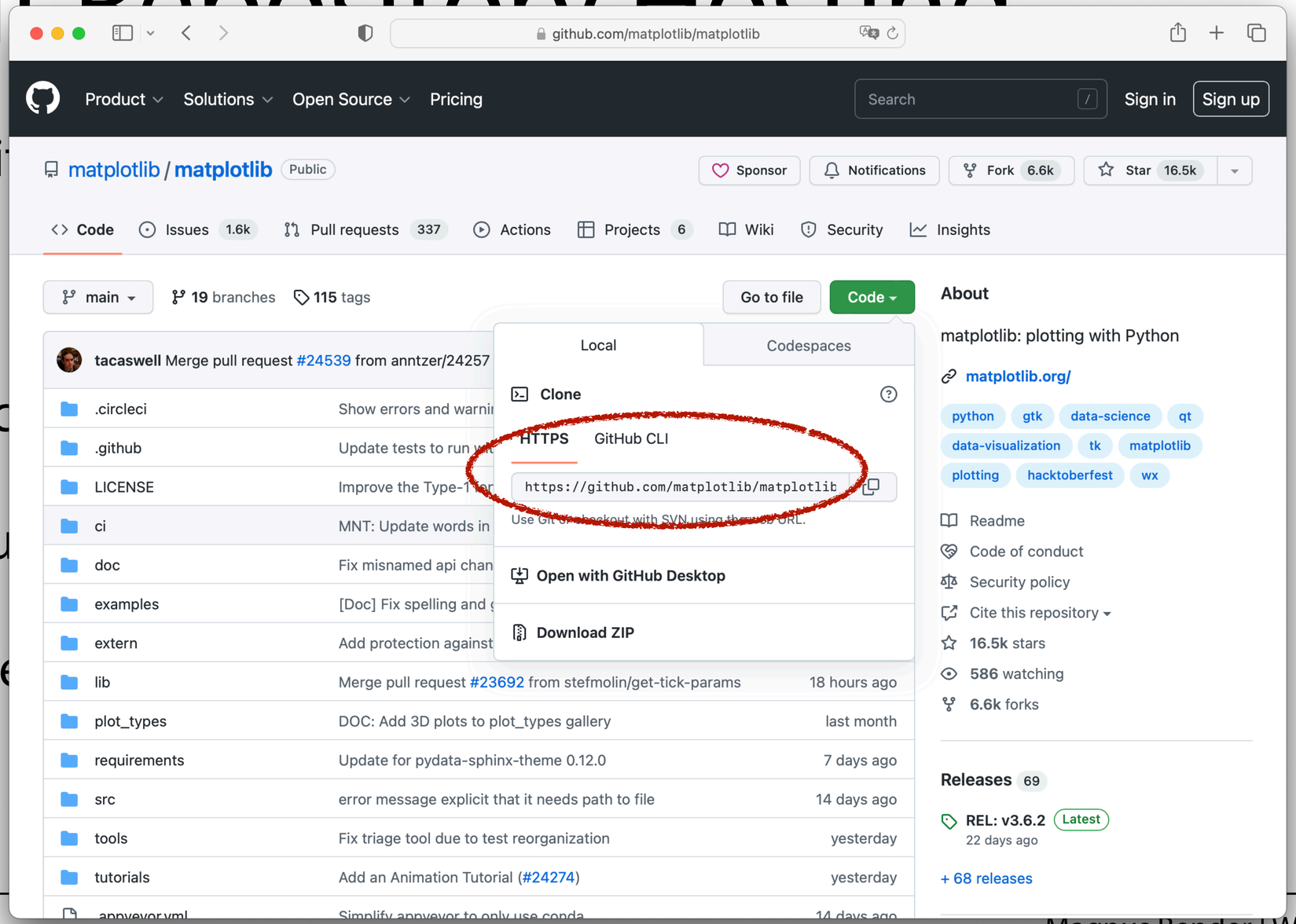
Git Repository Hosting

- GitHub, GitLab, Bi
- Webinterface zur
  - Commits, Branc
  - Forks, Pull Requ
- Projektmanageme
  - Issues, …



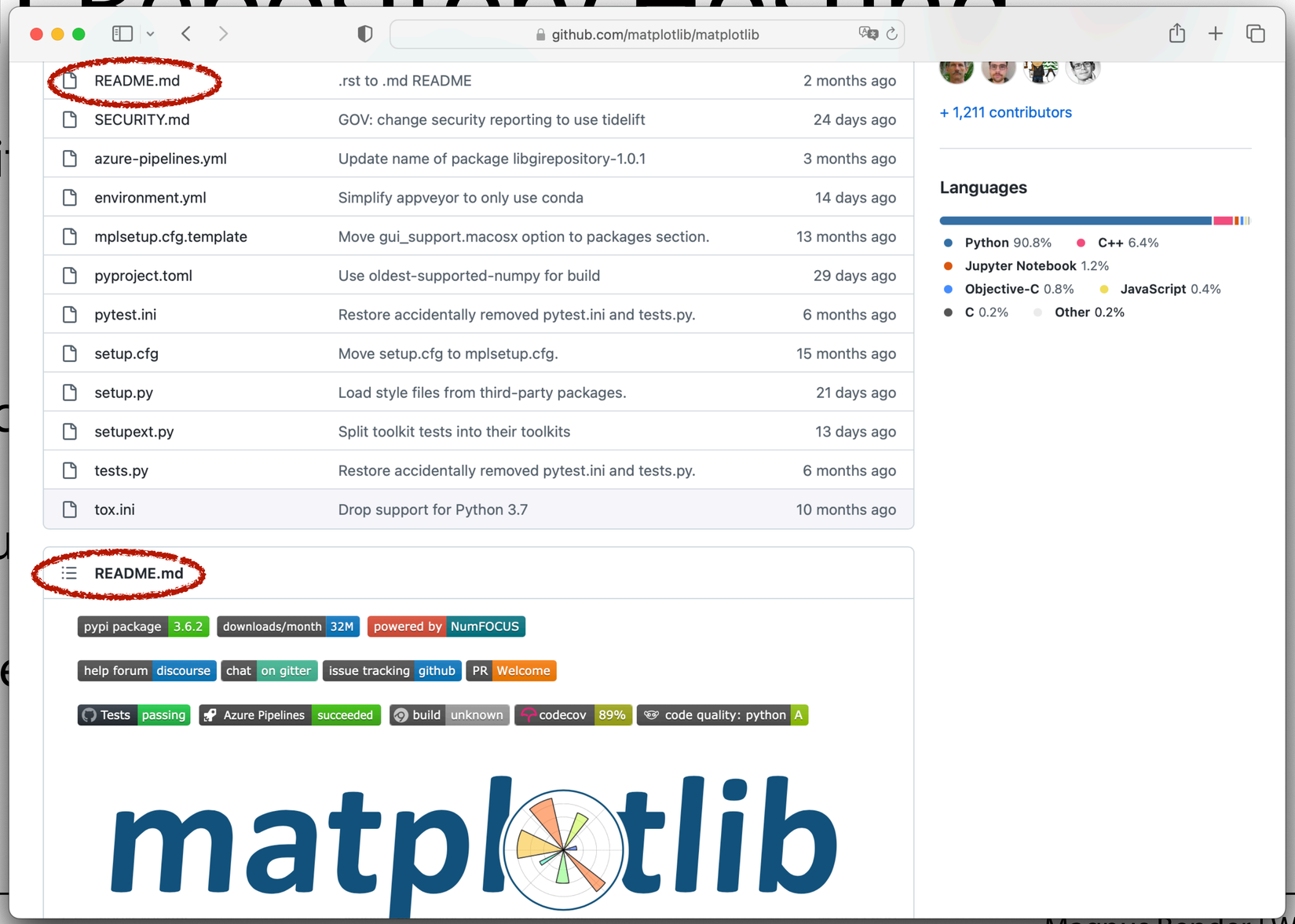
Git Repository Hosting

- GitHub, GitLab, Bi
- Webinterface zur
  - Commits, Branc
  - Forks, Pull Requ
- Projektmanageme
  - Issues, …



Git Repository Hosting

- GitHub, GitLab, Bi
- Webinterface zur
  - Commits, Branc
  - Forks, Pull Requ
- Projektmanageme
  - Issues, …



Zusammenfassung

I. Versionsverwaltung

II. Git

    1. Idee, Konfiguration

    2. Lokal: Commit, Stash, Branch, Merge

    3. Remote: Push, Pull

III. GitHub

Zusammenfassung

I. Versionsverwaltung

II. Git

    1. Idee, Konfiguration

    2. Lokal: Commit, Stash, Branch, Merge

    3. Remote: Push, Pull

III. GitHub

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) NumPy, SciPy
6. Datenverarbeitung und -visualisierung
   a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
   a) Grundlegende Ansätze (Datensätze, Auswertung)
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers