Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2023/24

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
- Verlauf soll gespeichert werden

`data.py`

`data_fixed.py`

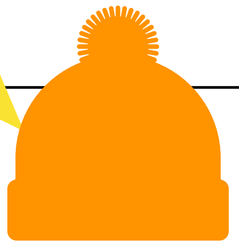
`20221123_data.py`

`plot.py`

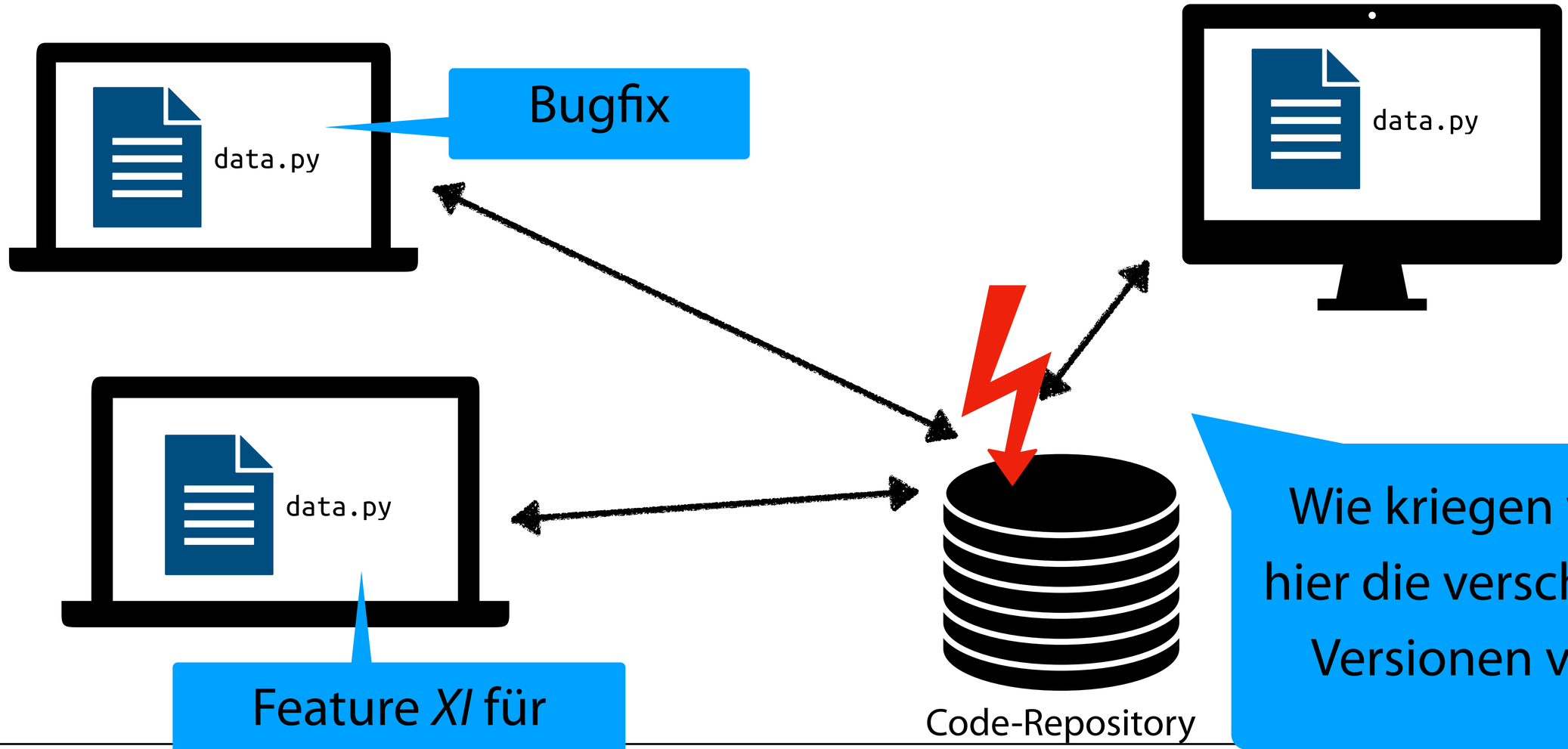
`plot_v2.py`

`20221001_plot.py`

`import` nutzt den Dateinamen :-(



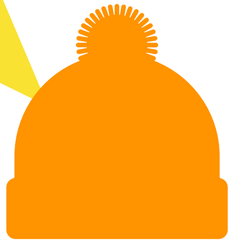
Verschiedene Entwicklungsorte



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!

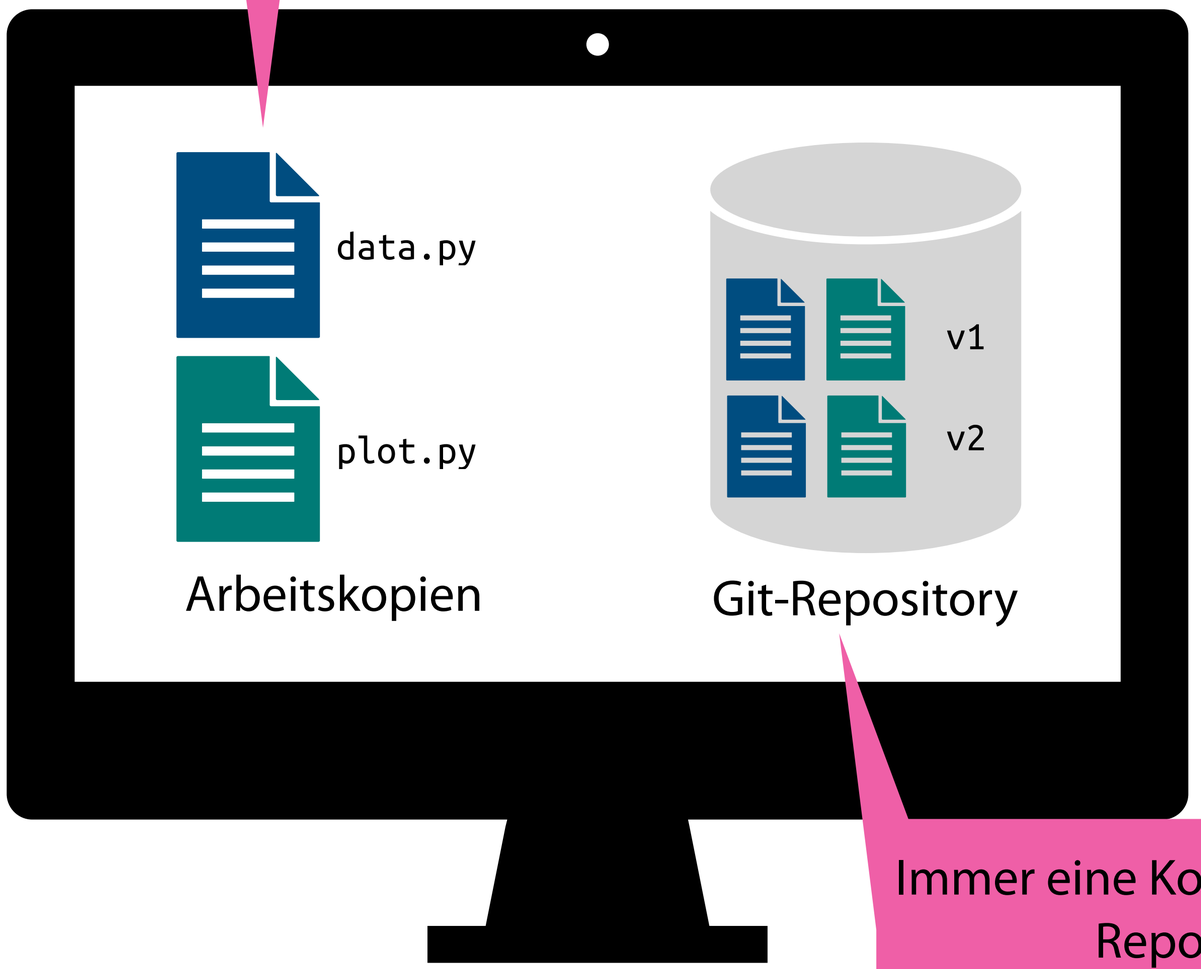


II.
Git

1. Idee, Konfiguration

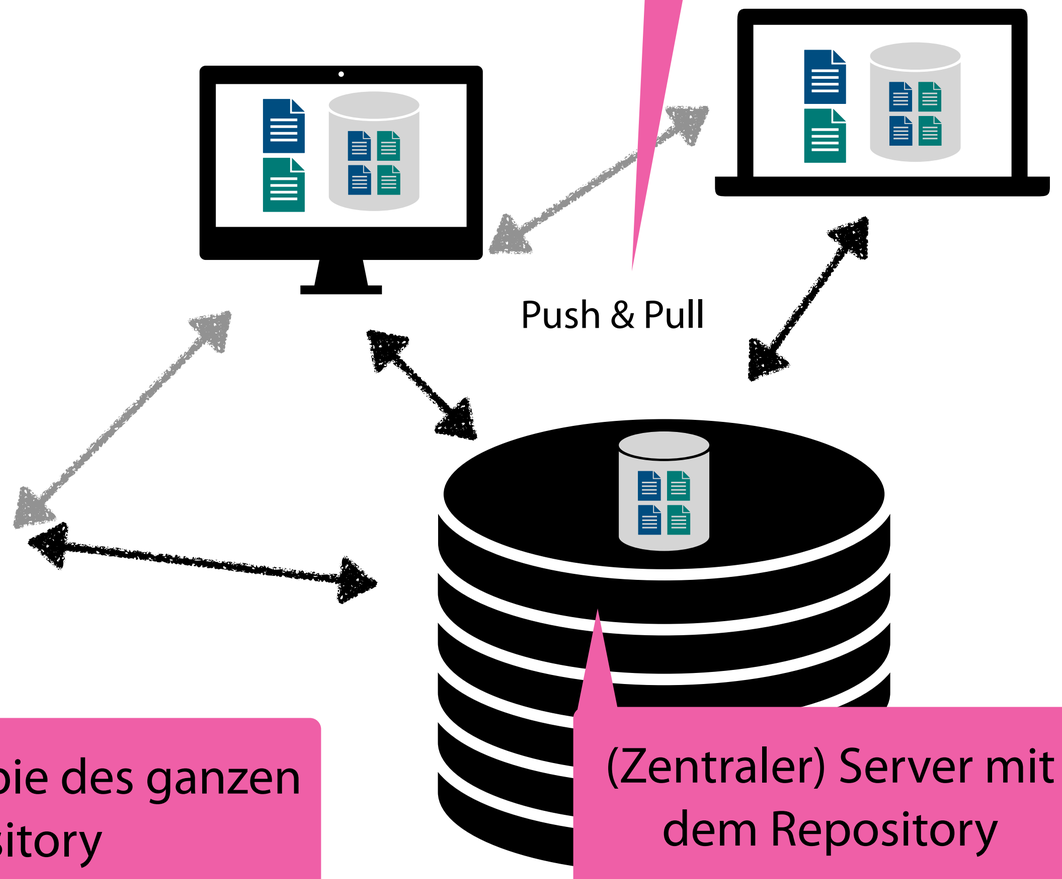
erteilte Repositories

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



Immer eine Kopie des ganzen Repository

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

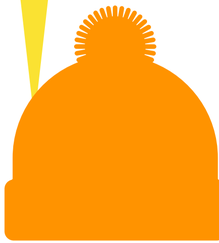


(Zentraler) Server mit dem Repository

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

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

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

Konfiguration

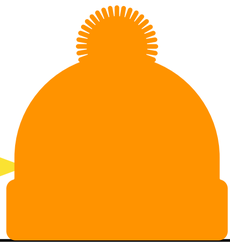
Gilt für ganzen Benutzeraccount, ohne `--global` für aktuelles Repo.

```
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
```


Git-Repository

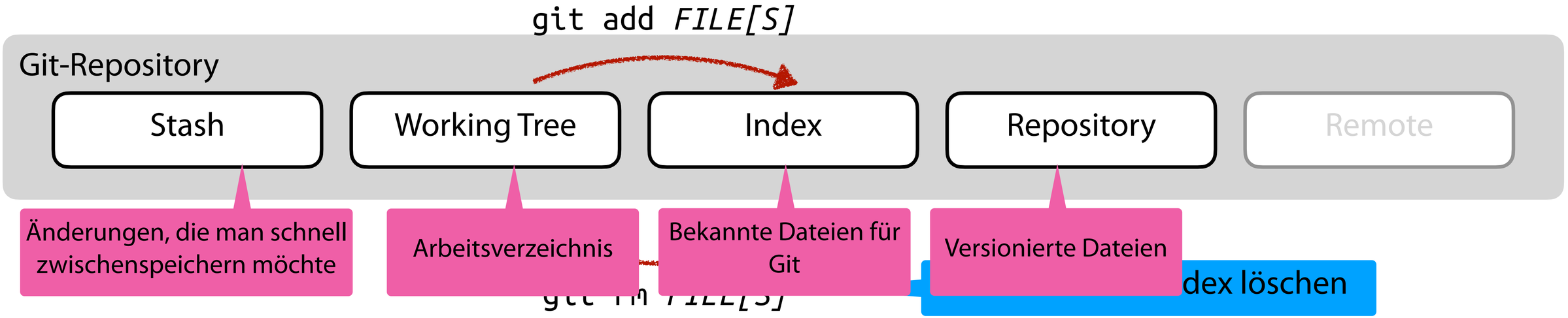
data.py

plot.py

Arbeitskopien

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

Dateien hinzufügen



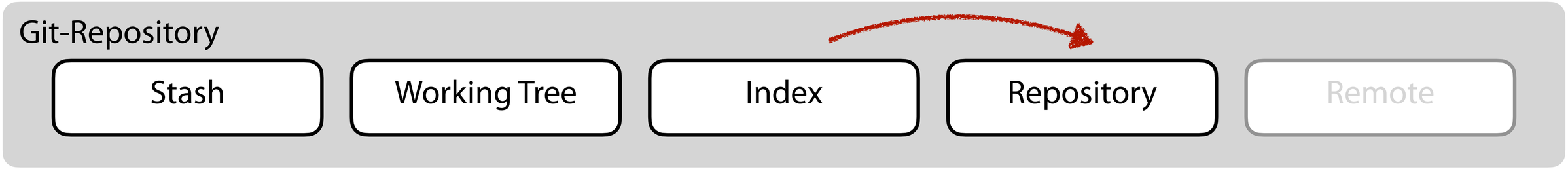

```
$> git status
On branch main
  No commits yet
$> touch a.txt
```

```
$> 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 commit [-m "Meine Änderung"]`

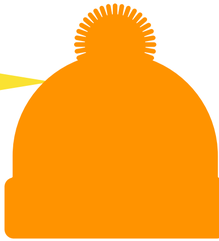


Kurz beschreiben, was getan wurde
(ohne `-m` öffnet sich ein Editor für mehr Text)

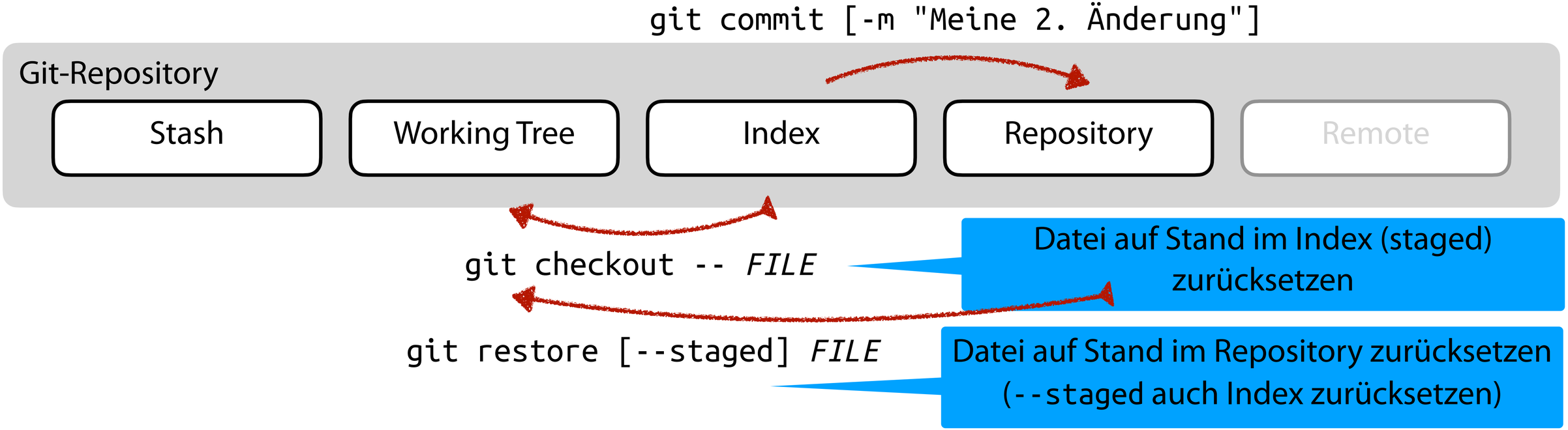
```
$> git status
  On branch main
    No commits yet
  Changes to be committed:
    new file:   a.txt
    new file:   b.txt
```

```
$> 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!



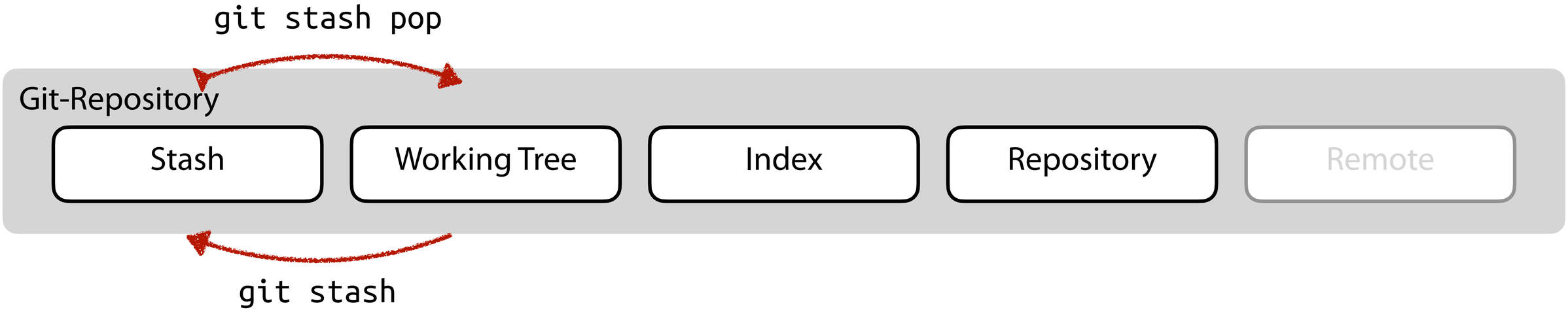
Weitere Änderungen




```
$> 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
```

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
```

```
bdeea00 (HEAD -> master, tag: v2.7.2, GH/master, GH/HEAD) Version 2.7.2
afb25b9 Merge pull request #29 from KIMB-technologies/im-export
7108562 (GH/im-export, im-export) Im- & Export works (#27)
439be50 Import & Replace works #27
3d0bb42 Im-Export Begin (#27)
bbadcaf (tag: v2.7.1) Version 2.7.1
8993f3e Merge pull request #28 from KIMB-technologies/more-configs
0f56e55 Assets Reload by Version, Toggle UnRead not for OwnStream
f65a5dc More Configs Cache & Log Dir.
4608720 (tag: v2.7.0) Version 2.7
84629d8 Merge pull request #25 from KIMB-technologies/no-docker
fb13725 (GH/no-docker, no-docker) Merge branch 'master' into no-docker
d806afa Version Hints #24
f69a560 More for Non-Docker mode (#23)
3adf0e6 No-Docker mode begin (#23)
36d398d (tag: v2.6.0) Version 2.6 & Fix Typo
ec3f9e5 Update Screenshots
545b008 Translation #19
d2ad4bc Remove "Previous Page" Button #20
3b3a2d7 Merge pull request #21 from KIMB-technologies/better-own-streams
2af496d (GH/better-own-streams, better-own-streams) Own Stream readme
daa6e75 More on #17, open: Test & Readme
b984a08 New Own Streams #17, Optim. Routing and Proxy
:
```

`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

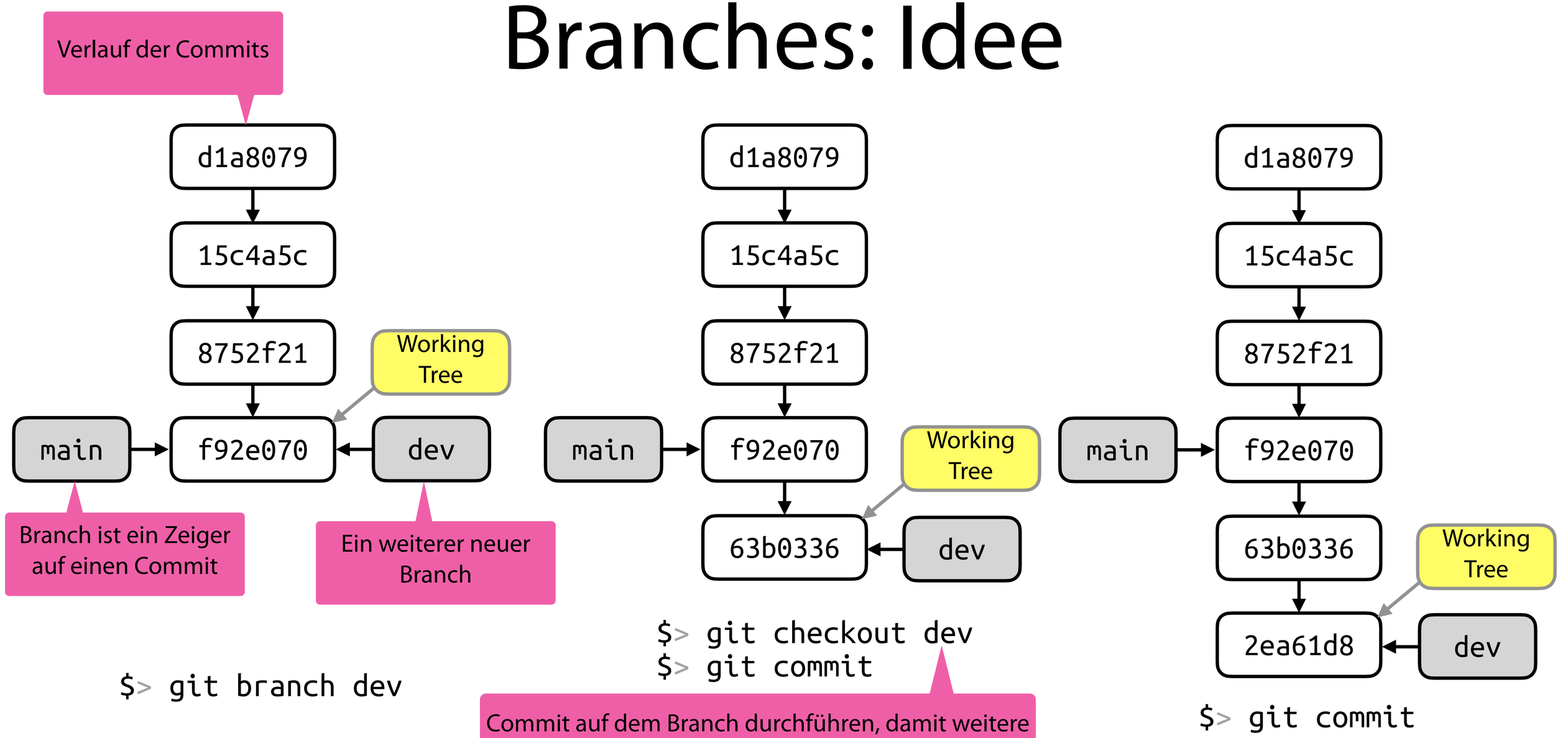
- [x] Verlauf der Änderungen (textbasierte Dateien)
- [ ] Verschiedene **Entwicklungszweige** gleichzeitig
  - Verschiedene (neue) Features und Fehlerbehebungen
  - Verschiedene Orte
- [ ] Zusammenführen von **Entwicklungszweigen**
- [x] Ein (zentrales) Repository

Branches

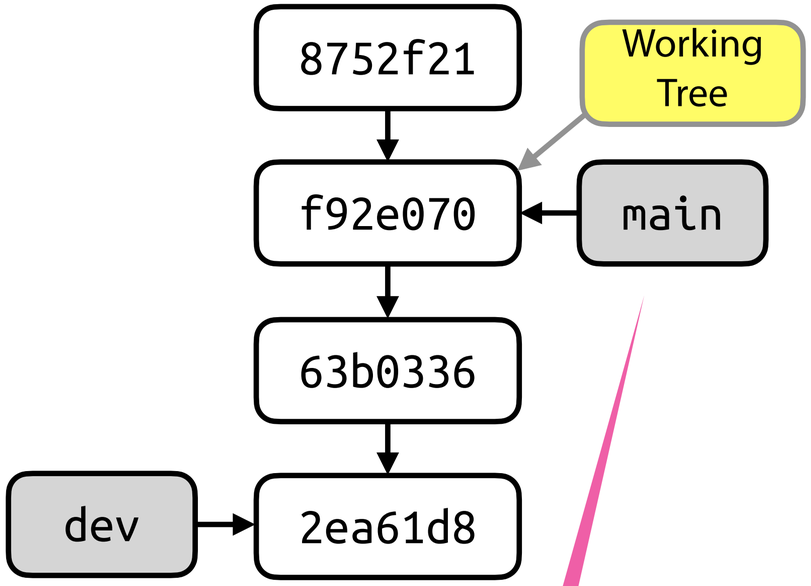
```
* bdeea00 (HEAD -> master, tag: v2.7.2, GH/master, GH/HEAD) Version 2.7.2
*   afb25b9 Merge pull request #29 from KIMB-technologies/im-export
|\
| * 7108562 (GH/im-export, im-export) Im- & Export works (#27)
| * 439be50 Import & Replace works #27
| * 3d0bb42 Im-Export Begin (#27)
|/
* bbadcaf (tag: v2.7.1) Version 2.7.1
*   8993f3e Merge pull request #28 from KIMB-technologies/more-configs
|\
| * 0f56e55 Assets Reload by Version, Toggle UnRead not for OwnStream
| * f65a5dc More Configs Cache & Log Dir.
|/
* 4608720 (tag: v2.7.0) Version 2.7
*   84629d8 Merge pull request #25 from KIMB-technologies/no-docker
|\
| *   fb13725 (GH/no-docker, no-docker) Merge branch 'master' into no-docker
| |\
```

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

Branches: Idee

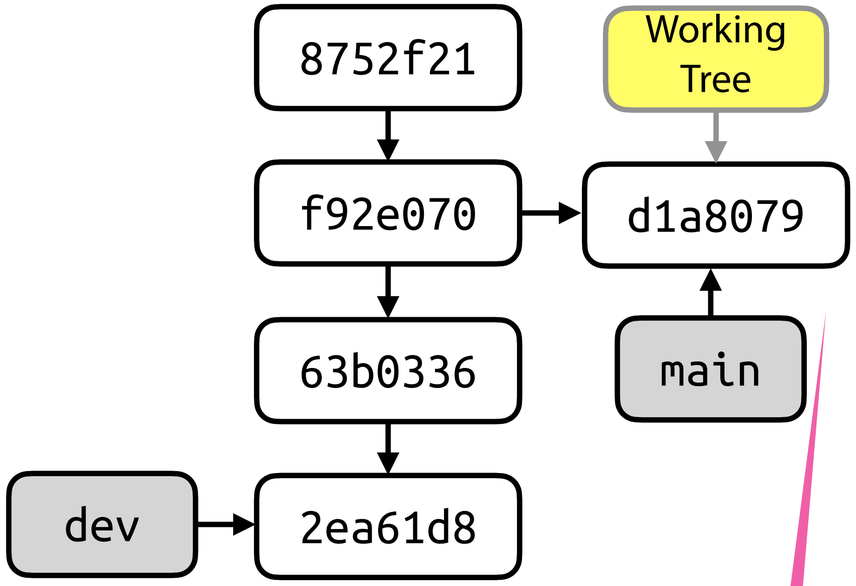


Branches: Verzweigung




```
$> git checkout main
```

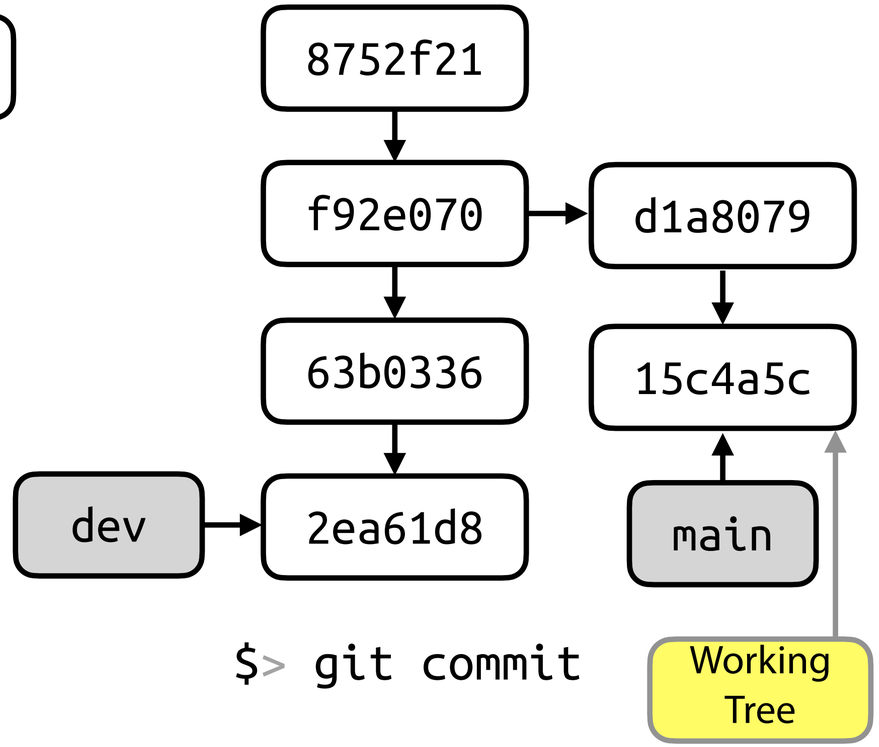
Auf einen anderen Branch wechseln




```
$> git commit
```

Was passiert jetzt?

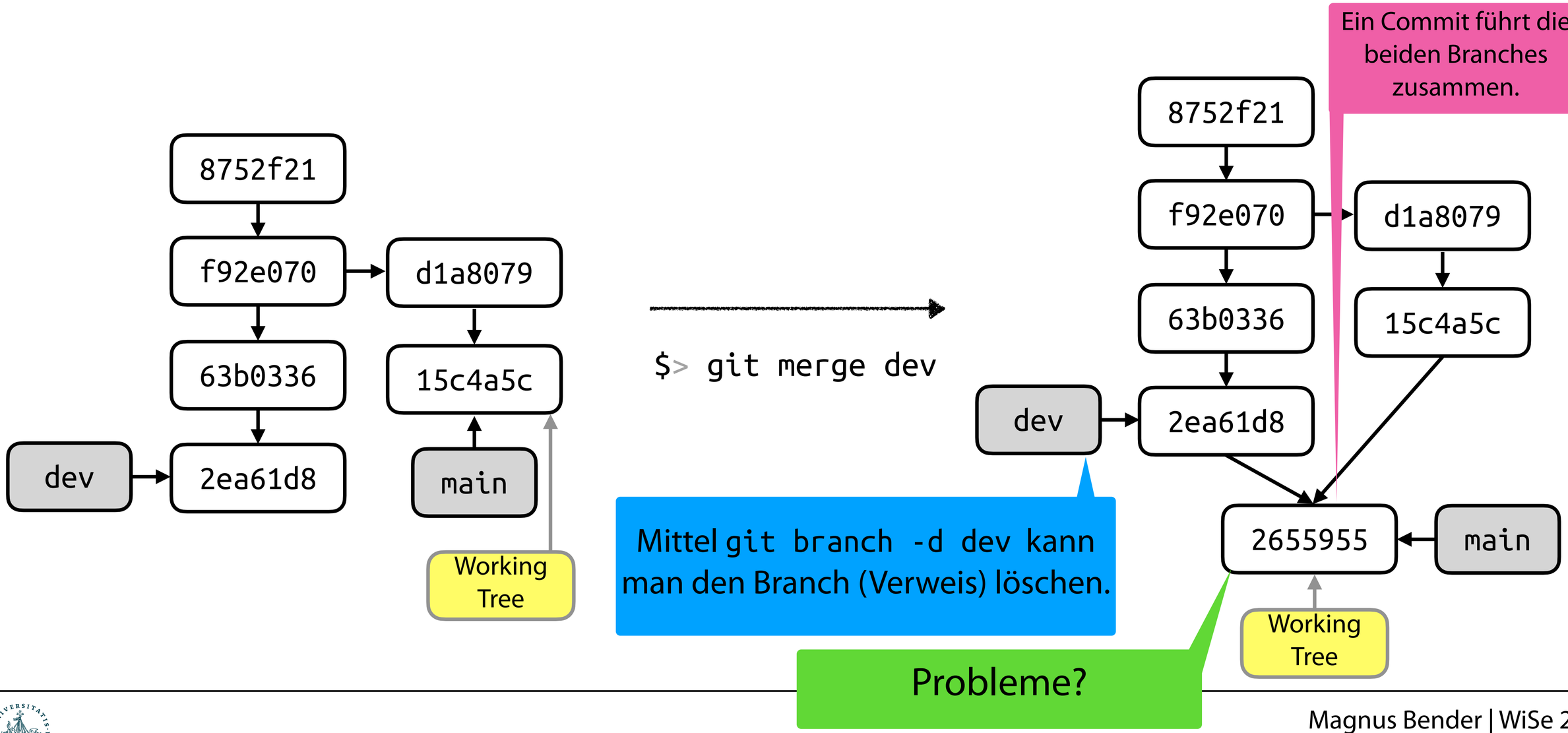
Auf dem anderen Branch Commits hinzufügen



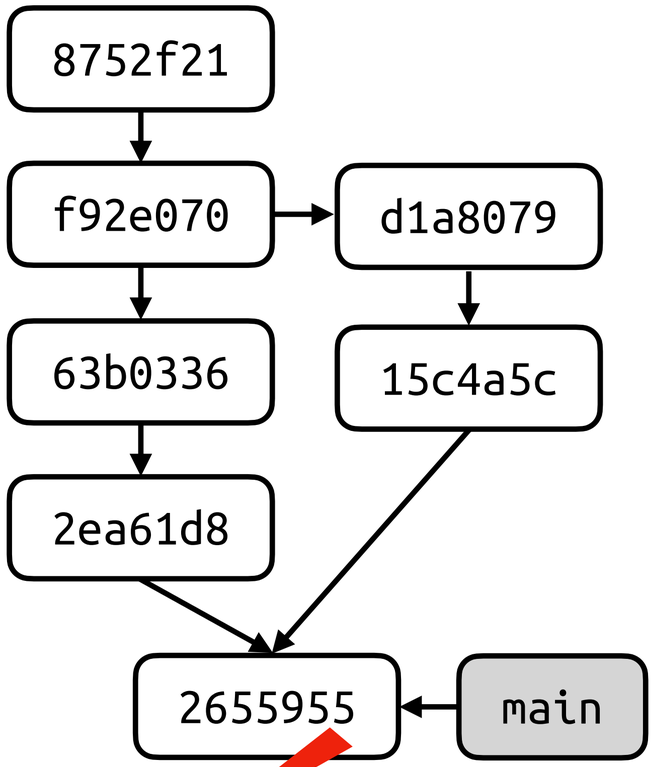

```
$> git commit
```

Working Tree

Branches: Zusammenführung



Zusammenführung: Konflikte

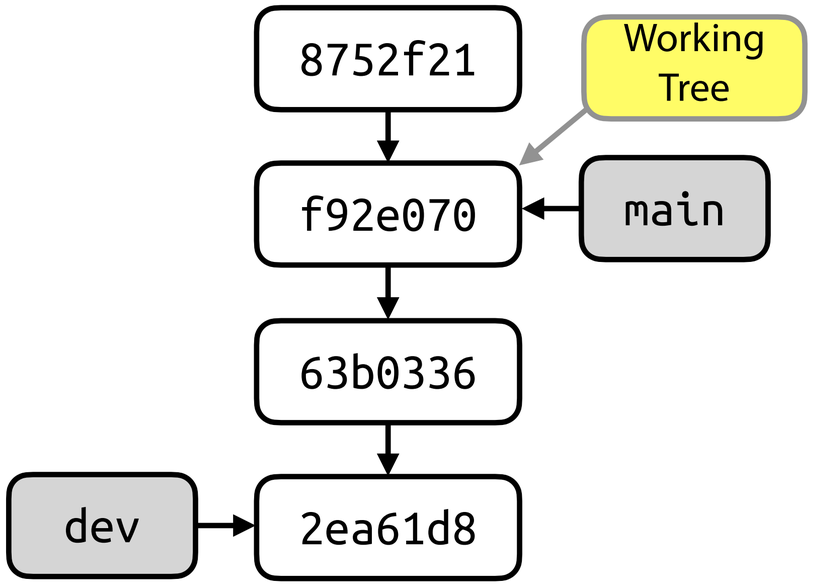


Merge-Konflikte treten auf, falls dieselbe Zeile der selben Datei in beiden Branches bearbeitet wurde.

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

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

$> git add DateiMitFehler.txt

$> git commit
```

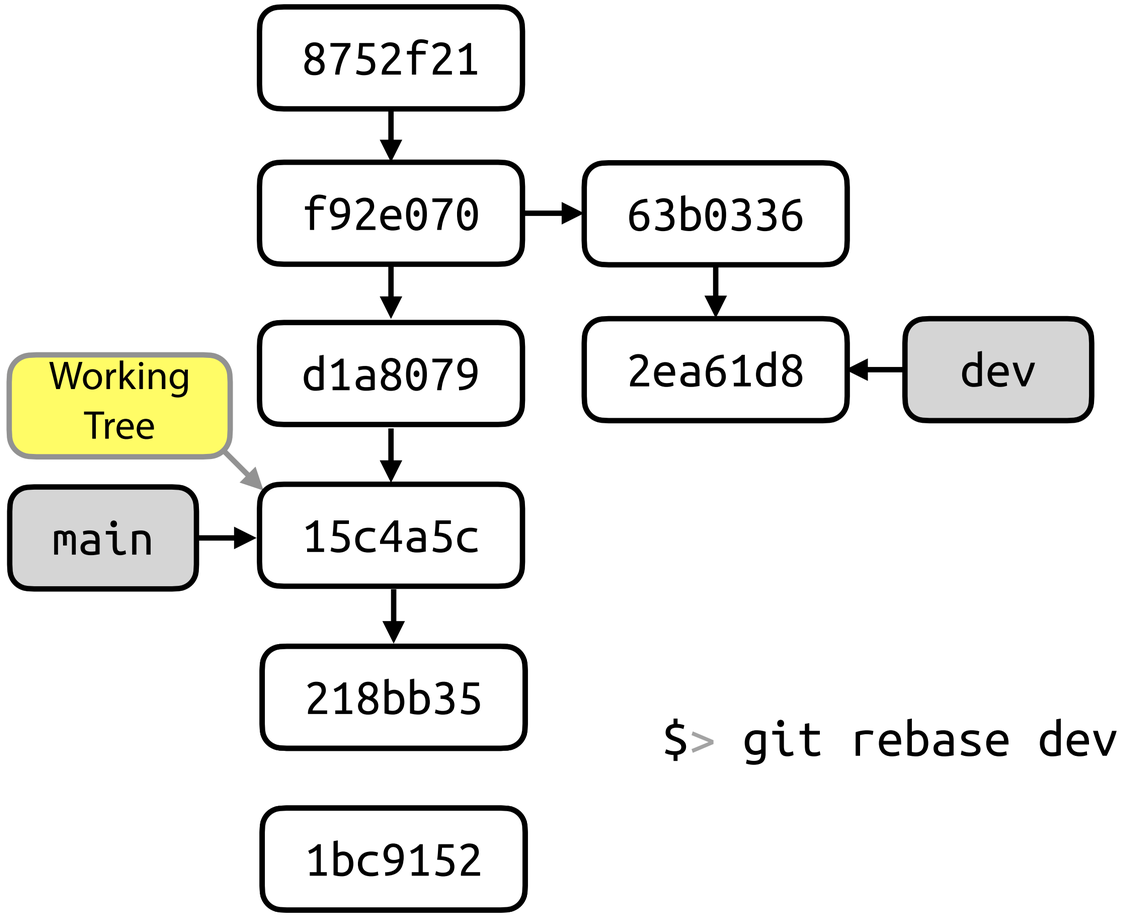
Weitere Zusammenführungen

Fast-Forward




```
$> git merge dev
```

Rebase




```
$> git rebase dev
```

„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

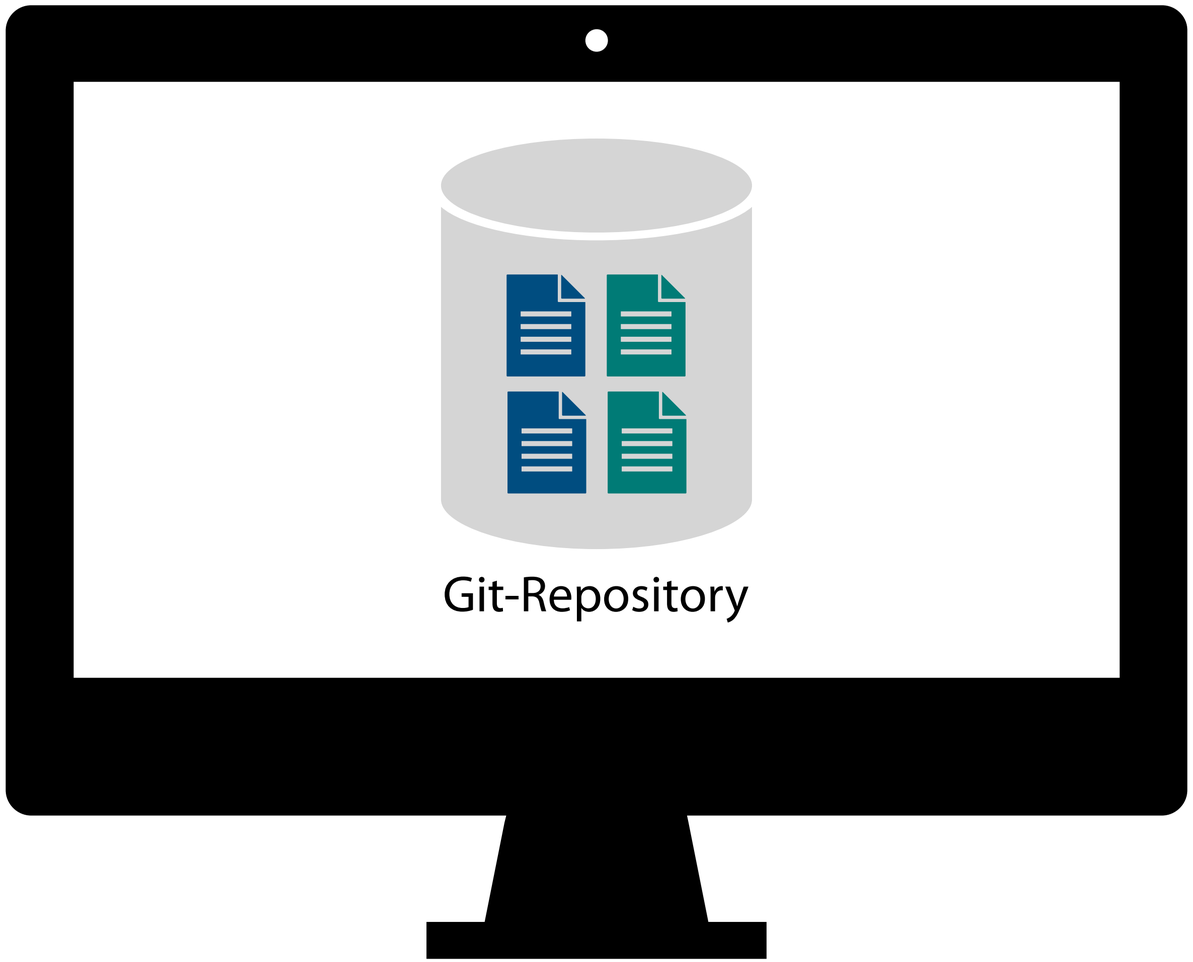
Naja, wir arbeiten immer im lokalen Repository!

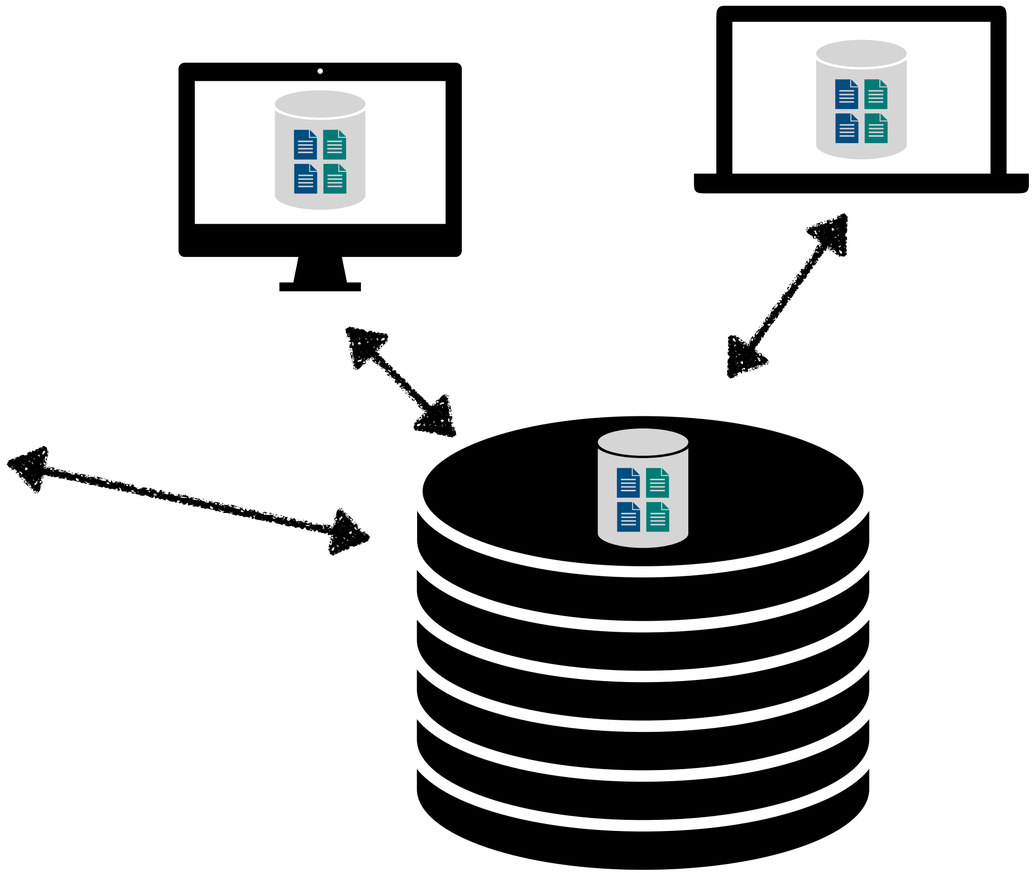
- ☑ Zusammenführen von **Entwicklungszweigen**
- ☑ Ein (zentrales) Repository

II.
Git

3. Remote: Push, Pull

Verteilte Repositories





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

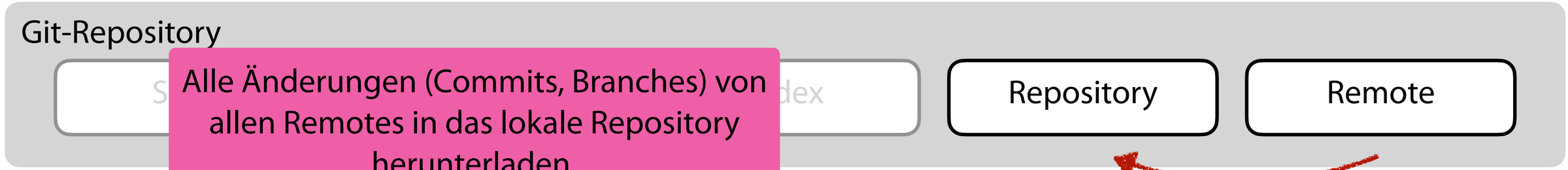
Quelle, üblicherweise benannt als `origin`
Schreiben i.A. nicht erlaubt

Server, Nutzer/Besitzer, Repository

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

Herunterladen

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
```

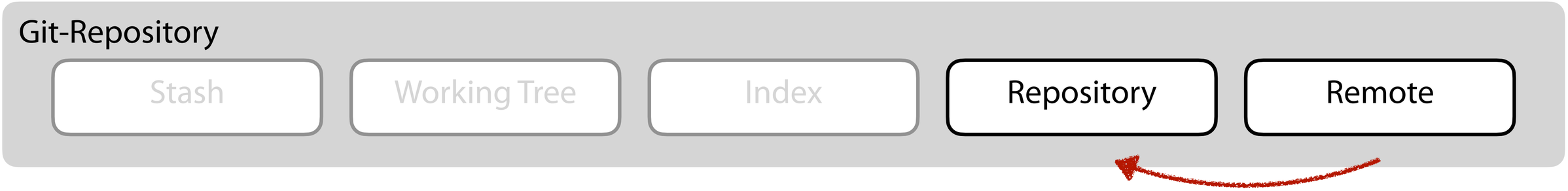
```
$> git fetch --all
  remote: Enumerating objects: 513, done.
  remote: Counting objects: 100% (116/116), done.
  remote: Compressing objects: 100% (107/107), done.
  remote: Total 513 (delta 48), reused 0 (delta 0), pack-reused 397
  Receiving objects: 100% (513/513), 271.85 KiB | 1.93 MiB/s, done.
  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
```

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

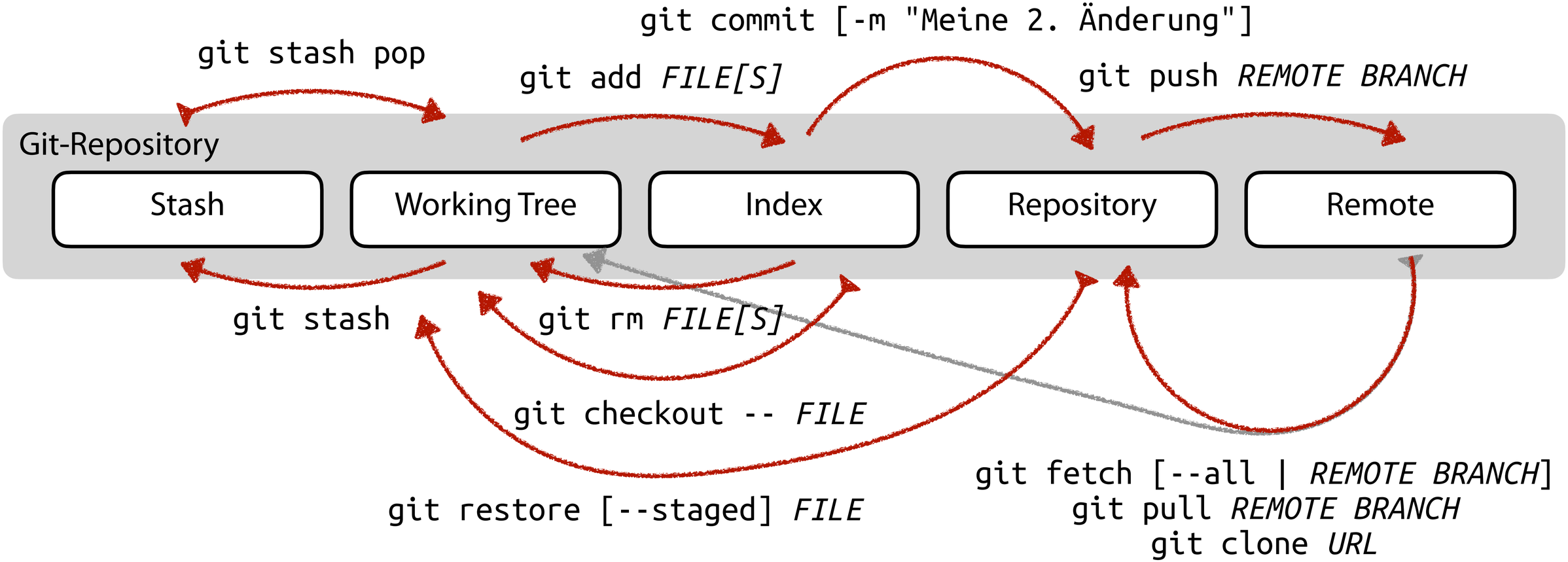
Hochladen


```
$> 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

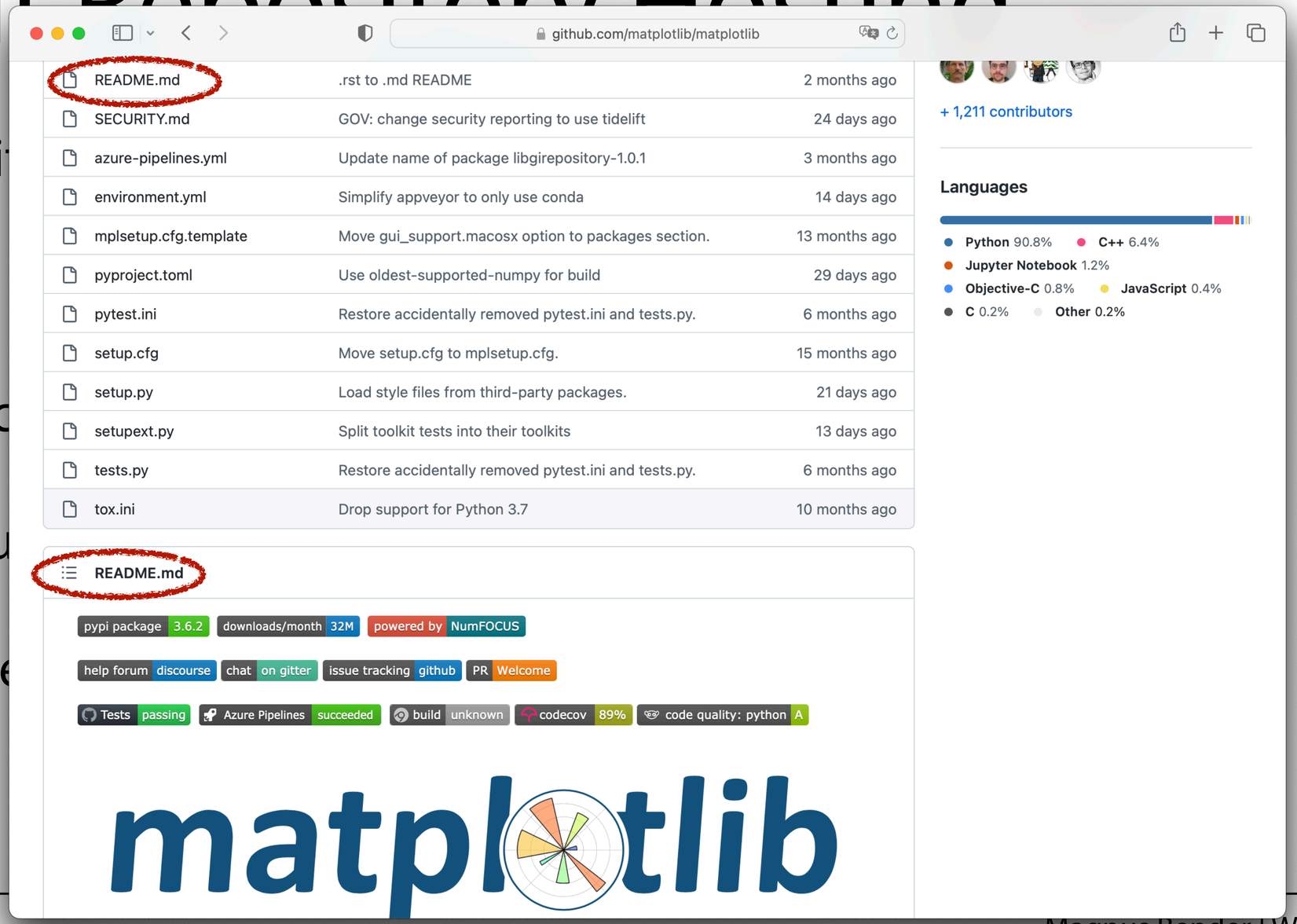
Git Befehle II



III.
GitHub

Git Repository Hosting

- GitHub, GitLab, 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

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