Grundlagen der Versionsverwaltung mit git und GitLab

Marius Politze, RWTH Aachen University

Einstieg ins Forschungsdatenmanagement mit git und GitLab

21.09.2021 12:00 - 17:00

Setting The Stage

Kurze Vorstellungsrunde

- Name, Institution, Tätigkeit
- Ein Punkt, den ich über `git` wissen möchte
- Eine HomeOffice-Beichte / -Anekdote
- ... das Wort weitergeben

Ergebnissicherung der Online-Vorbereitung

- Flipped Classroom Inhalte in GitLab

  https://gitlab-nrw-workshop-2021-09.gitlab.io/preparation/

- Rückfragen zu den Inhalten?

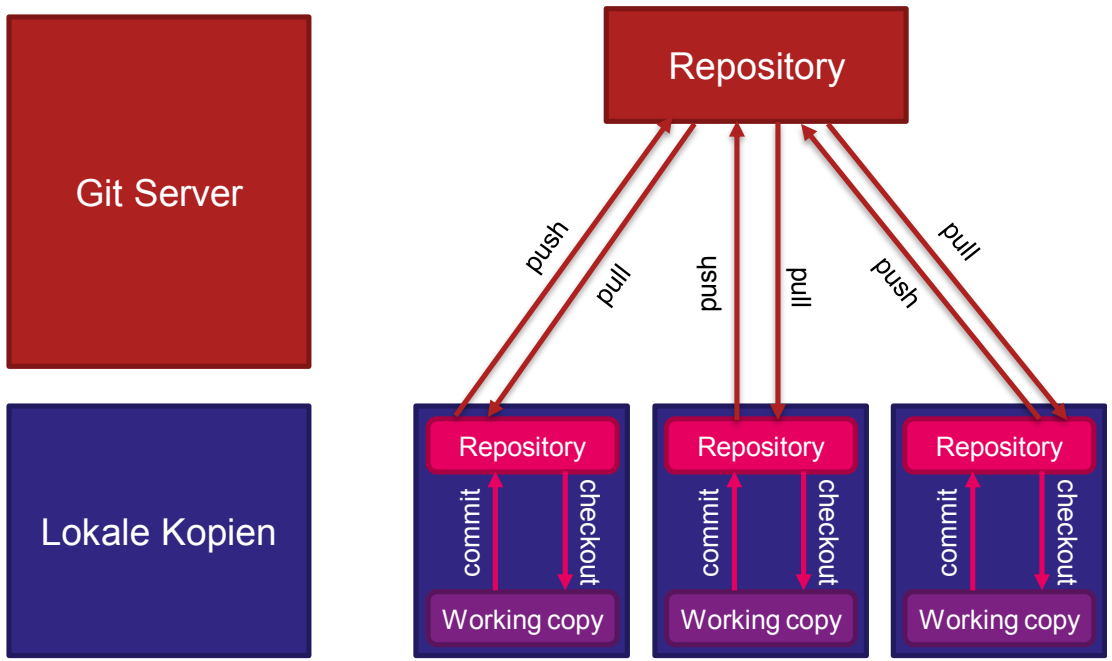
- Ideenspeicher für Fragen

  https://miro.com/app/board/o9J_l1ta3sw=/?moveToWidget=3074457362682111842

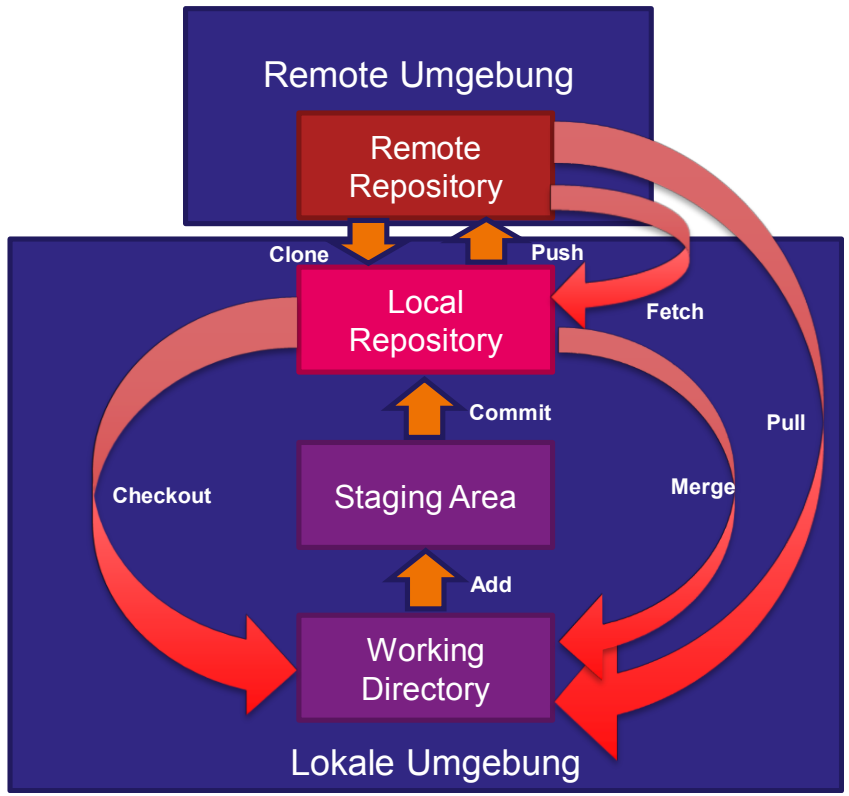
- Fehler bei der Installation? Abhilfe für Heute

  https://repl.it/@mpolitze/GitLab-NRW-Workshop

Git: Remote Version Control



Demo: Git Ablauf (2)



Visualizing Git Concepts with D3

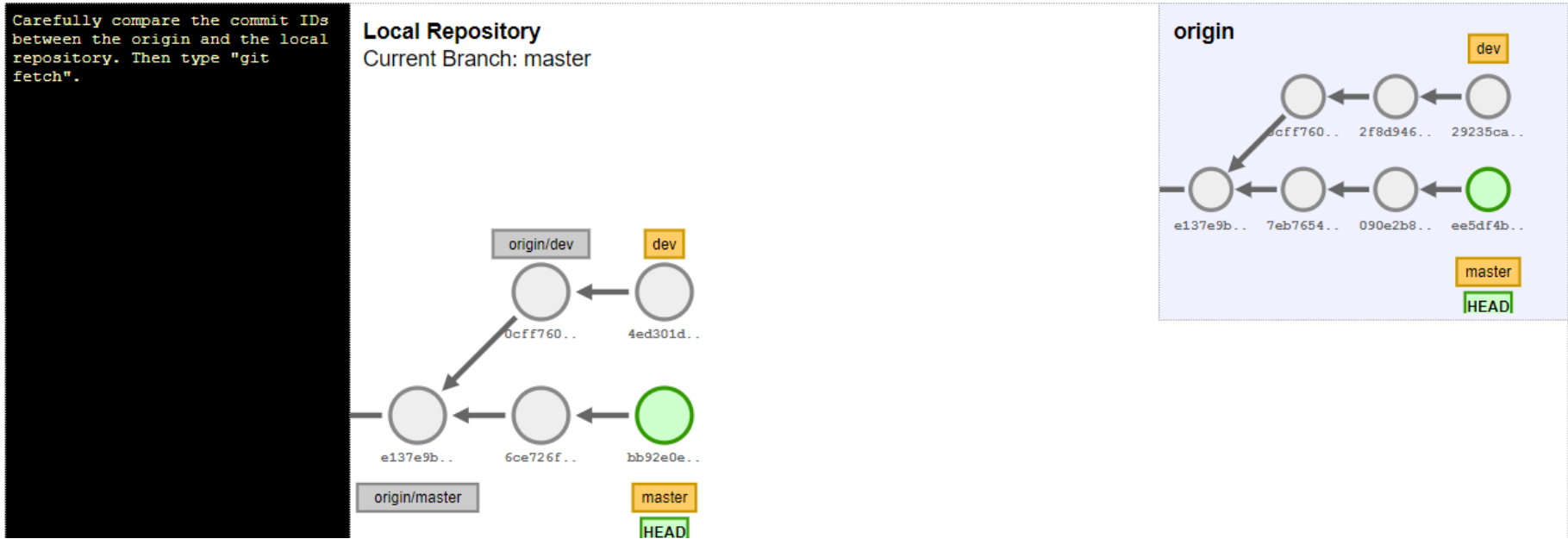
This website is designed to help you understand some basic git concepts visually. This is my first attempt at using both SVG and D3. I hope it is helpful to you.

Adding/staging your files for commit will not be covered by this site. In all sandbox playgrounds on this site, just pretend that you always have files staged and ready to commit at all times. If you need a refresher on how to add or stage files for commit, please read Git Basics.

Sandboxes are split by specific git commands, listed below.

Basic Commands		Undo Commits	Combine Branches	Remote Server	
git commit	git checkout	git reset	git merge	**git fetch**	git push
git branch	git checkout -b	git revert	git rebase	git pull	git tag

`git fetch` will update all of the "remote tracking branches" in your local repository. Remote tracking branches are tagged in grey.



Git Grundlegende Befehle - *Globale Konfiguration*

Konfiguration 1x pro Computer erforderlich:

Konfiguriert <Benutername> als globalen Git-Benutzernamen

```
git config --global user.name <Benutzername>
# z.B.
git config --global user.name "Marius Politze"
```

Konfiguriert <Email-Adresse> als globale Git-Emailadresse

```
git config --global user.email <Email-Adresse>
# z.B.
git config --global user.email "politze@itc.rwth-aachen.de"
```

Git Grundlegende Befehle - *Interaktion mit dem Repository*

Initialisiert im aktuellen Verzeichnis ein git Repository

```
git init
# z.B.
git init
```

Fügt Dateien dem staging Bereich hinzu

```
git add
# z.B.
git add .
```

Commitet die Dateien aus dem staging Bereich ins Repository

```
git commit -m <commit message>
#z.B.
git commit –m "Erster Commit"
```

Klont ein entferntes Repository

```
git clone <repository url>
# z.B.
git clone https://git.rwth-aachen.de/grp/repo.git
```

Versionshistorie an das entfernte Repository senden

```
git push <repository url> <branch>
# z.B.
git push https://git.rwth-aachen.de/grp/repo.git master
# oder den aktuellen branch zum default remote
git push
```

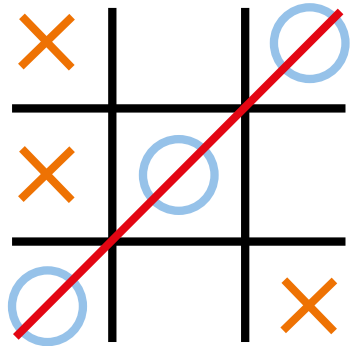
Holt den Versionsstand aus dem entfernten Repository und mergt in das Arbeitsverzeichnis

```
git pull <repository url> <branch>
# z.B.
git pull https://git.rwth-aachen.de/grp/repo.git master
# oder den aktuellen branch vom default remote
git pull
```

Übung: Git Tac Toe

Übung: Git Tac Toe - Regeln

- 2 Spieler: X und 0
- Auf einem Feld 3x3
- Spieler ziehen abwechselnd
- In jedem Zug setzt der Spieler sein Symbol in ein Feld
- Gewonnen hat, wer drei Symbole in einer Zeile, Spalte oder Diagonale hat



In einer Textdatei sieht das Spielfeld dann so aus:

```
 | |
-+-+-
 | |
-+-+-
 | |
```

Übung: Git Tac Toe - *Runde 1 Abwechselnd (1)*

Vorbereitung: Teilen Sie sich in 2er Gruppen mit X und 0

X erstellt ein GitLab Repository und initialisiert das Projekt mit einer Readme.md

X lädt 0 als *Maintainer* in das Projekt ein

Vorbereitung: Beide Spieler klonen das Repository auf ihren Rechnern

Vorbereitung: X legt eine `Runde1.txt` mit dem Spielfeld an

```
X> git add Runde1.txt
X> git commit -m "Spielfeld für Runde 1"
```

Übung: Git Tac Toe - *Runde 1 Abweschselnd (2)*

Spielzug: X macht den ersten Zug im Spielfeld

```
X> git add Runde1.txt
X> git commit -m "X Zug 1"
X> git push
```

Spielzug: 0 aktualisiert das Repository und macht den nächsten Zug

```
O> git pull
```

0 macht den Zug im Spielfeld

```
O> git add Runde1.txt
O> git commit -m "O Zug 1"
O> git push
```

Spielzug: X ist an der Reihe und startet mit git pull und dem zweiten Zug

... und so weiter ...

Übung: Git Tac Toe - *Runde 2 Gleichzeitig (1)*

Vorbereitung: X erstellt eine neue Datei `Runde2.txt`, füllt diese mit dem Spielfeld

```
X> git add Runde2.txt
X> git commit -m "Spielfeld für Runde 2"
X> git push
```

Spielzug gleichzeitig: X und 0 machen ihre Züge

```
X&O> git pull
```

Zug im Spielfeld eintragen

```
X&0> git add Runde2.txt
X&O> git commit -m "{X,O} Zug 1"
```

Übung: Git Tac Toe - *Runde 2 Gleichzeitig (2)*

Spielzug nacheinander: Änderungen an den Server senden (Erst Spieler 0!)

```
X&O> git push
```

Der push von X wird rejected!

X muss den Konflikt lösen:

```
X> git pull
```

Entweder: git macht einen automatischen merge Oder: X öffnet `Runde2.txt` und führt die Spielstände zusammen.

```
X> git add Runde2.txt
X> git commit -m "Zug 1 zusammengeführt"
X> git push
```

... und so weiter ...

Machen Sie noch eine Runde 3 mit getauschten Rollen X und 0

Übung: Git Tac Toe - *Runde 3 Branches (1)*

Vorbereitung: X erstellt eine neue Datei `Runde3.txt` und füllt diese mit dem Spielfeld

```
X> git add Runde4.txt
X> git commit -m "Spielfeld für Runde 3"
X> git push
```

Vorbereitung: 0 aktualisiert das Repository:

```
O> git pull
```

Vorbereitung: Beide erstellen einen Branch für ihre Spielzüge:

```
X&O> git checkout -b "r4s{X,O}"
X&O> git push --set-upstream origin r4s{X,O}
```

X und 0 spielen jeweils auf dem eigenen Branch.

Übung: Git Tac Toe - *Runde 3 Branches (2)*

Spielzug gleichzeitig: X und 0 machen den Spielzug und senden Änderungen an den Server

Branches aktualisieren

```
X&O> git pull
```

Zug im Spielfeld eintragen

```
X&O> git add Runde4.txt
X&O> git commit -m "{X,O} Zug 1"
X&O> git push
```

Übung: Git Tac Toe - *Runde 3 Branches (3)*

Spielzug X: Führt die Änderungen zusammen

```
git fetch
git merge origin/r4s0
```

X führt Spielstände zusammen

```
git add Runde4.txt
git commit -m "Zug 1 zusammengeführt"
git push
```

Spielzug 0: Holt Änderungen von X

```
git fetch
git merge origin/r4sX
```

... und so weiter ...

Machen Sie noch eine Runde mit getauschten Rollen X und 0

Git Cheat Sheet

Remember!
`git <COMMAND> --help`

Global configuration is stored in ~/.gitconfig.
`git config --help`

master is the default development branch.
origin is the default upstream repository.

Create

From existing data
```
cd ~/my_project_directory
git init
git add .
```
From existing repository
```
git clone ~/existing_repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://user@host.org/project.git
```

Show

Files changed in working directory
`git status`

Changes made to tracked files
`git diff`

What changed between ID1 and ID2
`git diff <ID1> <ID2>`

History of changes
`git log`

History of changes for file with diffs
`git log -p <FILE> <DIRECTORY>`

Who changed what and when in a file
`git blame <FILE>`

A commit identified by ID
`git show <ID>`

A specific file from a specific ID
`git show <ID>:<FILE>`

All local branches
`git branch`
star (*) marks the current branch

Revert

Return to the last commited state
`git reset --hard`
This cannot be undone!

Revert the last commit
`git revert HEAD`
Creates a new commit

Revert specific commit
`git revert <ID>`
Creates a new commit

Fix the last commit
`git commit -a --amend`
(after editing the broken files)

Checkout the ID version of a file
`git checkout <ID> <FILE>`

Update

Fetch latest changes from origin
`git fetch`
(this does not merge them)

Pull latest changes from origin
`git pull`
(does a fetch followed by a merge)

Apply a patch that someone sent you
`git am -3 patch.mbox`
In case of conflict, resolve the conflict and
`git am --resolved`

Publish

Commit all your local changes
`git commit -a`

Prepare a patch for other developers
`git format-patch origin`

Push changes to origin
`git push`

Make a version or milestone
`git tag v1.0`

Branch

Switch to a branch
`git checkout <BRANCH>`

Merge BRANCH1 into BRANCH2
```
git checkout <BRANCH2>
git merge <BRANCH1>
```

Create branch BRANCH based on HEAD
`git branch <BRANCH>`

Create branch BRANCH based on OTHER and switch to it
`git checkout -b <BRANCH> <OTHER>`

Delete branch BRANCH
`git branche -d <BRANCH>`

Resolve merge conflicts

View merge conflicts
`git diff`

View merge conflicts against base file
`git diff --base <FILE>`

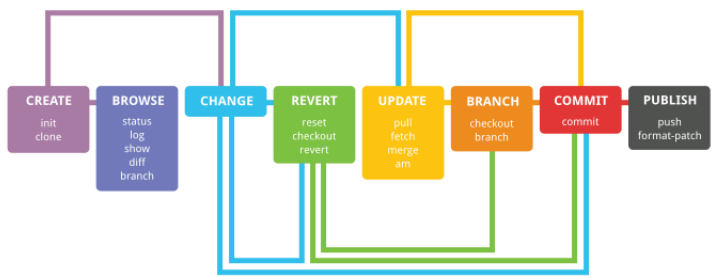
View merge conflicts against your changes
`git diff --ours <FILE>`

View merge conflicts against other changes
`git diff --theirs <FILE>`

Discard a conflicting patch
```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge with
```
git add <CONFLICTING_FILE>
git rebase --continue
```

Workflow



GitLab Walkthrough: *Wiki*

Zweck: Dokumentation der Daten im Repository

- Sind selbst ein Repository
- Seiten können verlinkt werden
- Markdown ermöglicht Formatierung
- Bilder, Videos und Quellcode können eingefügt werden

GitLab Walkthrough: *Wiki*

Zweck: Dokumentation der Daten im Repository

- Sind selbst ein Repository
- Seiten können verlinkt werden
- Markdown ermöglicht Formatierung
- Bilder, Videos und Quellcode können eingefügt werden

Demo: Wikis in GitLab

- Erstellen von Seiten
- Bearbeiten von Seiten
- Markdown:
  - Überschriften
  - Formatierung
  - Links
  - Listen
  - Quellcode
  - Bilder und Videos

GitLab Walthrough: *Issues*

Zweck: Projektmanagement direkt im Repository

- Einzelne Arbeitspakete
- Lassen sich Projektmitgliedern zuweisen
- Lassen sich zu Meilensteinen zusammenfassen
- Verschiedene Ansichten

GitLab Walthrough: *Issues*

Zweck: Projektmanagement direkt im Repository

- Einzelne Arbeitspakete
- Lassen sich Projektmitgliedern zuweisen
- Lassen sich zu Meilensteinen zusammenfassen
- Verschiedene Ansichten

Demo: Issues in GitLab

- Erstellen von Issues
  - Beschreibung
  - Labels
  - Metadaten
- Arbeitspakete Planen
  - Mit dem Meilensteinen
  - Mit Boards Mit dem Repository verknüpfen
  - Mit Commits
  - Mit Branches

Wrap Up

- Ideenspeicher

  https://miro.com/app/board/o9J_l1ta3sw=/?moveToWidget=3074457362682111842

- Fragen?

- Feedback Seestern

  https://miro.com/app/board/o9J_l1ta3sw=/?moveToWidget=3074457362682111898