

# Versionsverwaltung

## 4

# REMOTES

# GIT

Stupid Content Tracker, effizienter Objekt-Speicher mit der Fähigkeit zwei Objektspeicher miteinander zu Synchronisieren.

Synchronisation bei Daten == Replikation

---

Synchronisation der Git-Objekte == Replikation aller Git-Objekte

# REMOTES

- Das letzte verbliebene Element in der Liste der Git-Objekte
- Ein lokales **Repository** kann mit vielen verschiedenen **entfernten Repositories** verbunden sein
- wenn Repo durch **git clone** entstanden ist, gibt es genau ein Remote mit dem Namen **origin**

```
## Zeigt alle konfigurierten Remotes an
$ git remote -v
origin    https://github.com/barclay-reg/dhbw-slides.git (fetch)
origin    https://github.com/barclay-reg/dhbw-slides.git (push)
```

# BRANCHES SYNCHRONISIEREN

- Sync bei GIT-Objekten relativ einfach
- Sync bei Referenzen etwas komplexer
  - Problem: Referenz wurde auf beiden Seiten verändert
  - Lösung:
    - Speichern aller lokalen und remote Referenzen
    - Speichern der Verbindung zwischen lokaler und remote Referenz

# REMOTE REFERENZEN

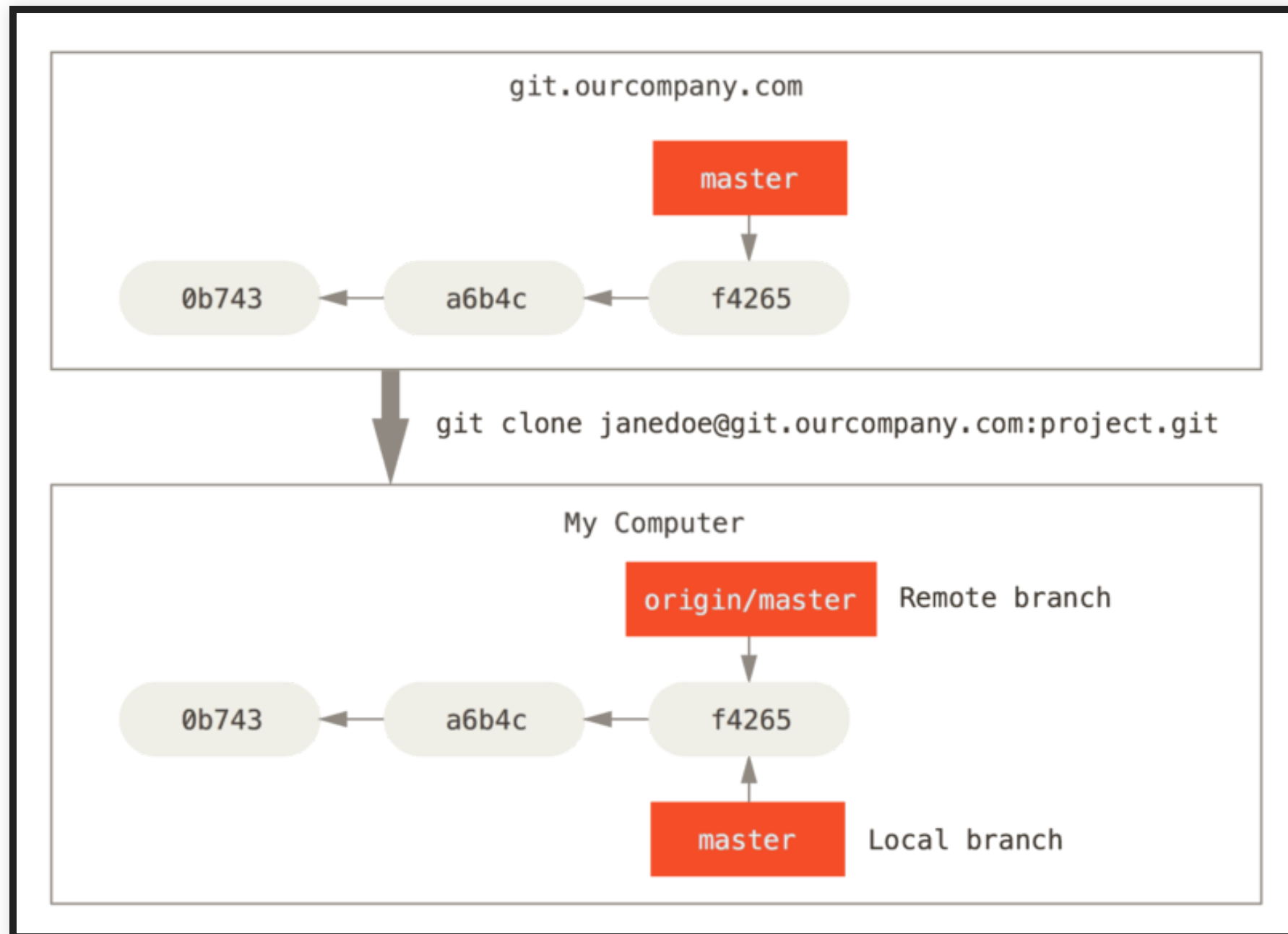
```
## Auflistung aller Dateien im Ordner .git/refs
$ find .git/refs
.git/refs
.git/refs/heads
.git/refs/heads/master
.git/refs/heads/my-branch-1
.git/refs/tags
.git/refs/tags/test-tag-0
.git/refs/tags/test-tag-1
.git/refs/remotes
.git/refs/remotes/origin
.git/refs/remotes/origin/master
.git/refs/remotes/origin/other-branch-2
```

# REMOTE TRACKING BRANCHES

- Wenn ein **lokaler** Branch mit einem **remote** Branch verbunden ist, dann spricht man von einem sog.:
  - **Remote-Tracking-Branch**
  - oder Upstream-Branch
  - Git weiss, dass der lokale Branch dem remote Branch *folgt*
  - z.B. der Branch `origin/master` wird von `master` getracked
- Name der remote Branches == Namen des Remote Repository plus dem Namen des Branches zusammen
  - z.B. `origin/feature-1`



# REMOTE TRACKING BRANCHES

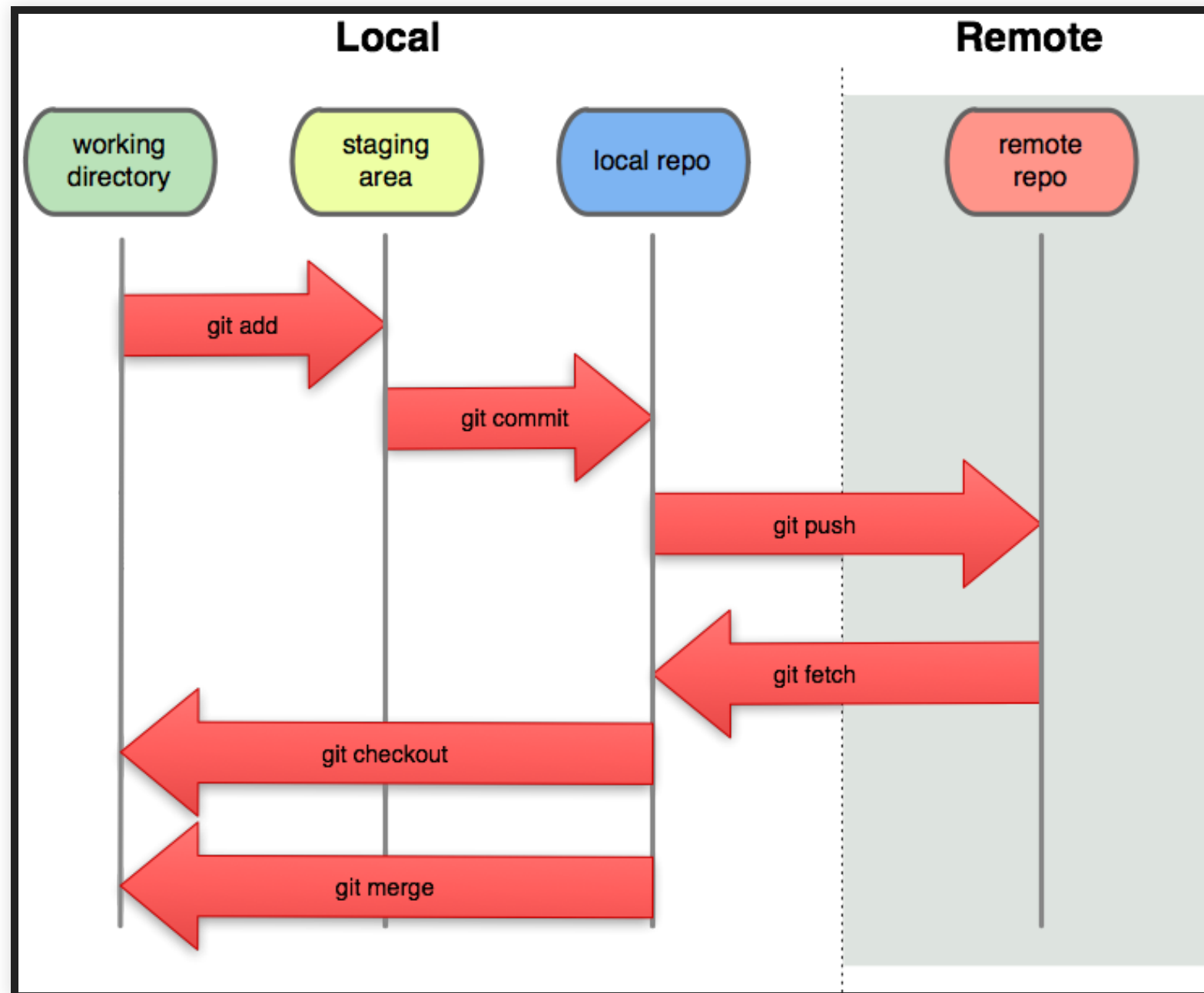


# KOMMANDOS

```
## Zeigt alle Remote Branches an
$ git branch -r
origin/master
origin/feature-1
## Legt neuen Branch an, der origin/feature-1 trackt
$ git branch feature-1 origin/feature-1
## Shortcut, legt automatisch lokalen Branch an, falls ein
## Branch `origin/feature-1` existiert
$ git checkout feature-1
## Definiert für den aktuellen lokalen Branch den
## Namen des Remote Branches
$ git branch --set-upstream-to origin/neues-feature
```

# ARBEITEN MIT REMOTES

# TRANSPORTWEGE

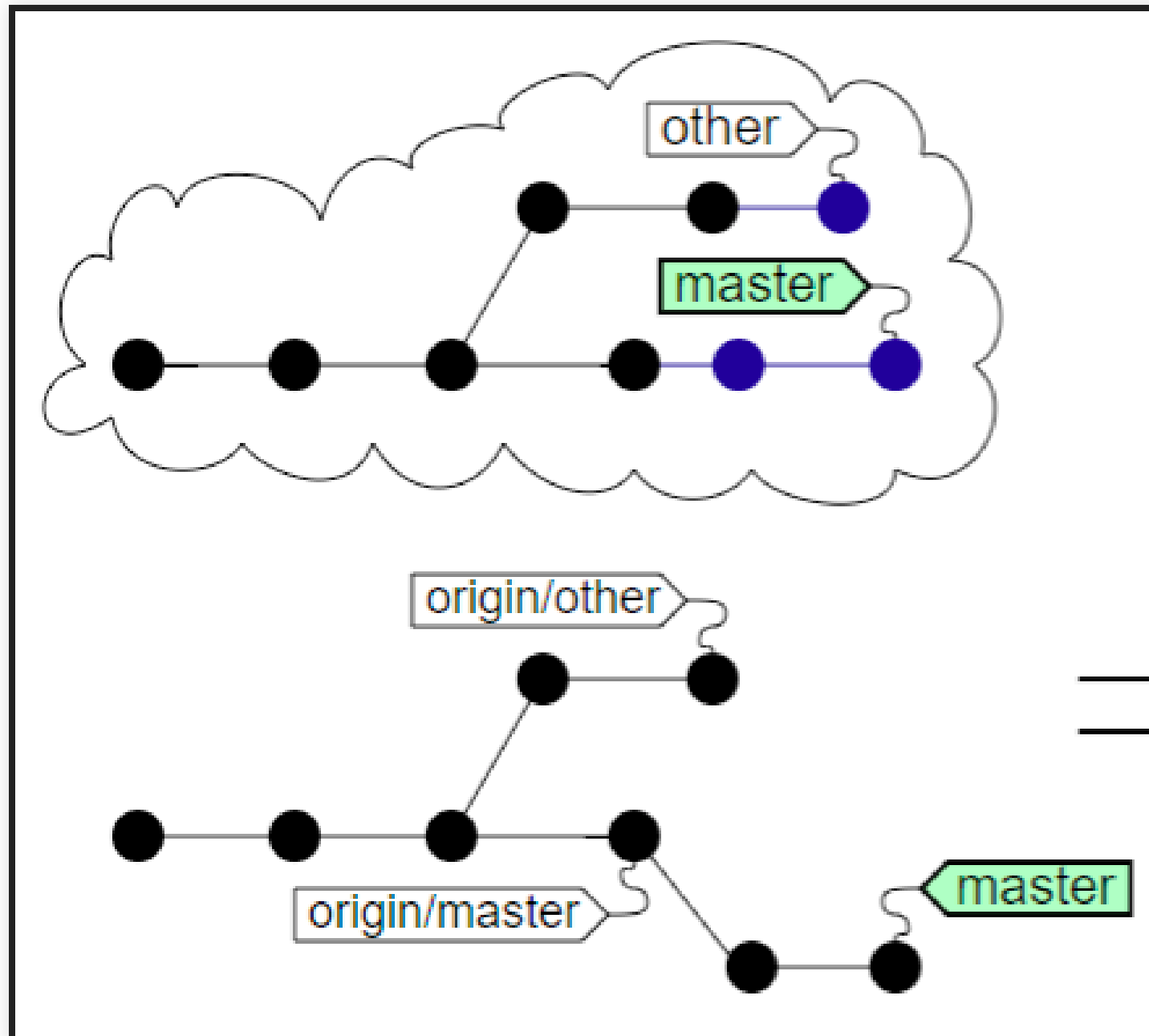


# FETCH UND PULL

- `git fetch`
  - Holt alle Commits von den remote Repository(ies)
  - keine Änderung an der Workcopy
  - oft ist danach ein Merge notwendig (3WM oder FF)
- `git pull`
  - Shortcut für `git fetch` & `git merge`

```
## ohne die Angabe von `origin` werden alle Remotes abgefragt
$ git fetch origin
## merge des Remote-Branche auf den aktuellen lokalen Branch
$ git merge origin/master
## Shortcut für die beiden oberen Befehle
## auch hier kann `origin` weggelassen werden
$ git pull origin
```

# FETCH UND PULL



# FETCH UND PULL

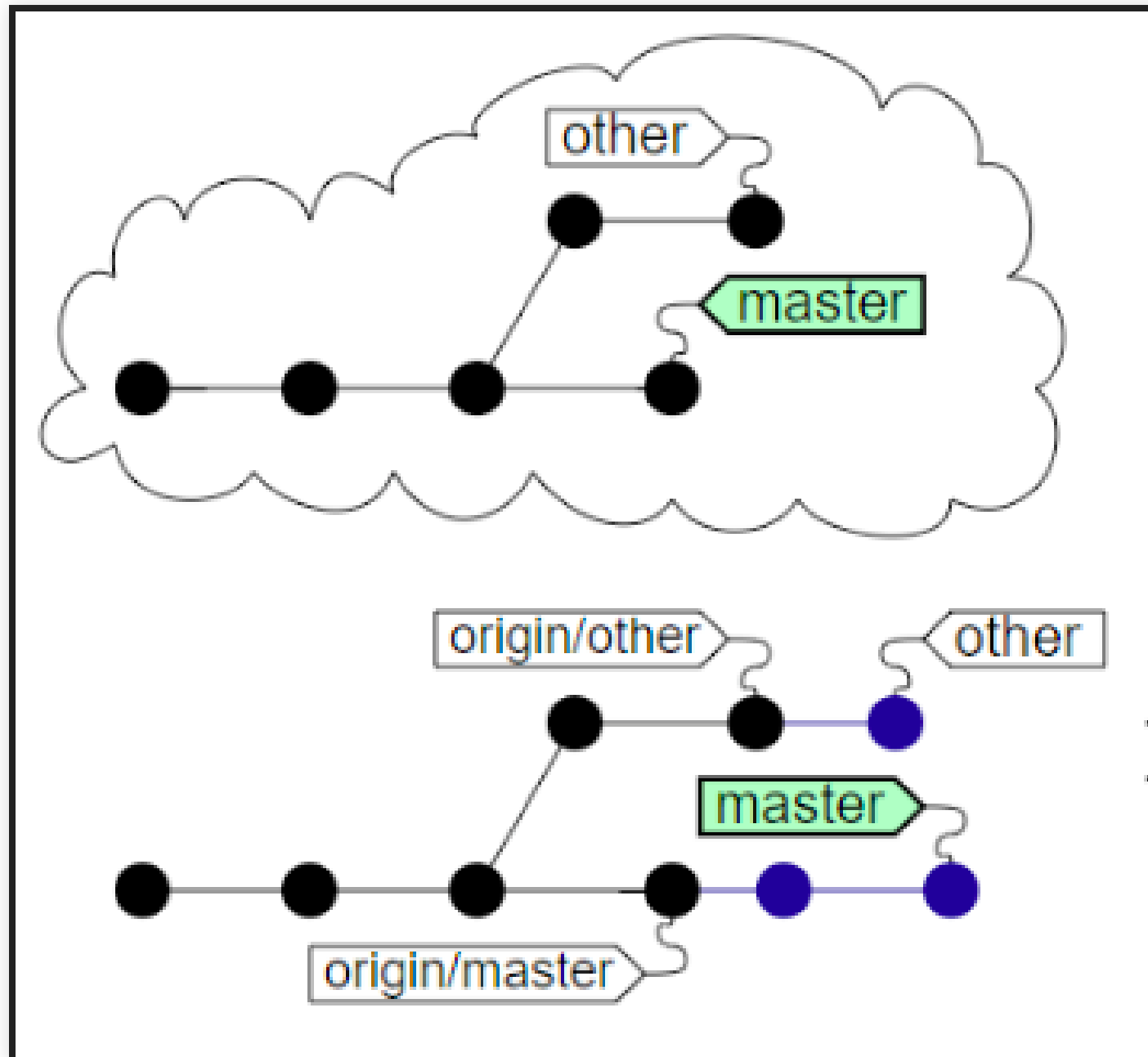


# PUSH

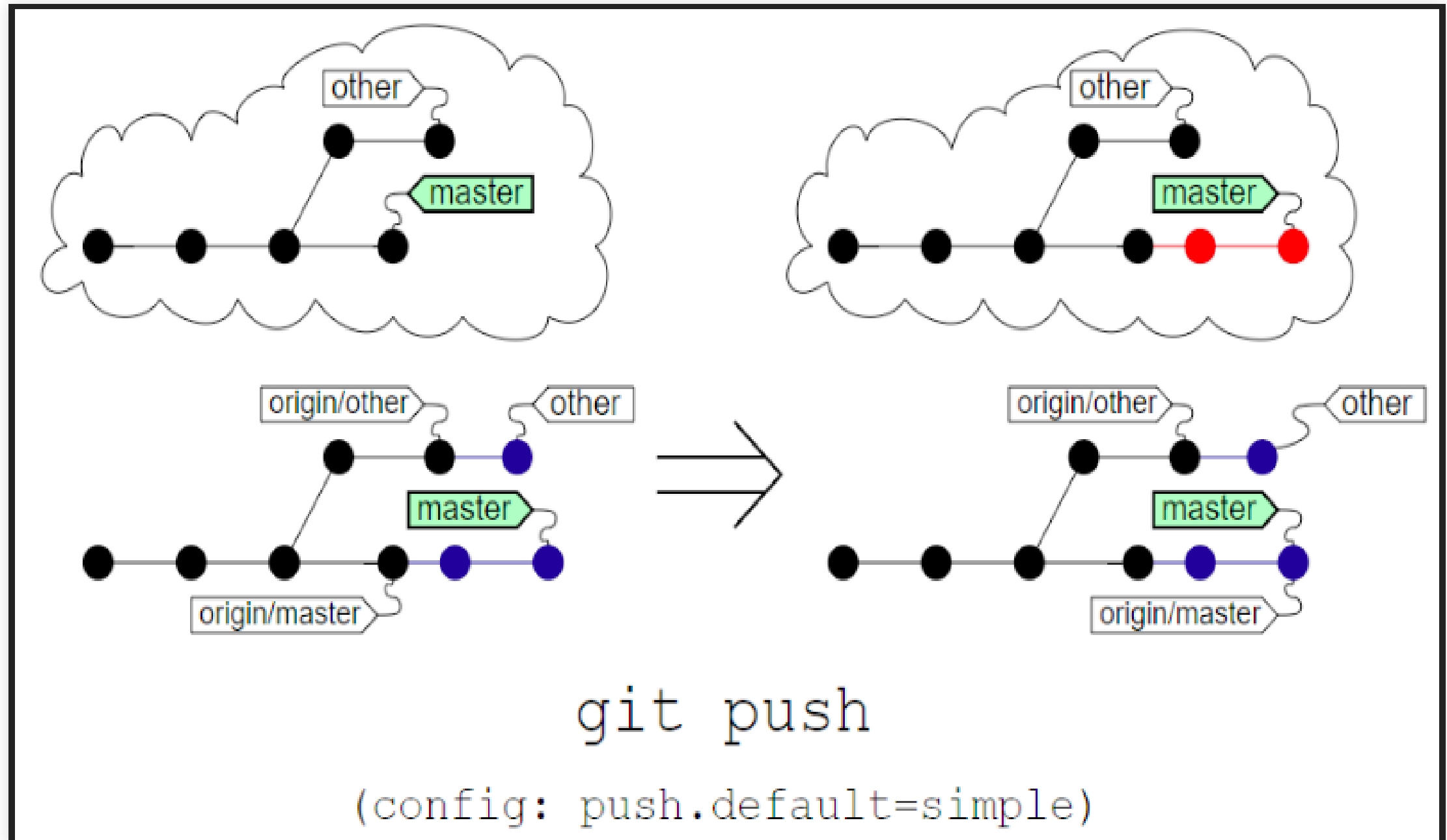
- `git push`
  - überträgt alle lokalen Commits zu dem Remote Repository
  - Nur erlaubt, wenn (remote) ein **Fast-Forward-Merge** möglich ist, ansonsten vorher `git pull`
  - danach ist **KEIN Ändern** der Historie/Commits empfohlen
    - Kein Commit-Amend, Reset von Branches, Rebasing
  - je nach Konfiguration wird nur der lokale Branch oder alle Branches synchronisiert
    - `config: push.default=simple`



# PUSH



# PUSH



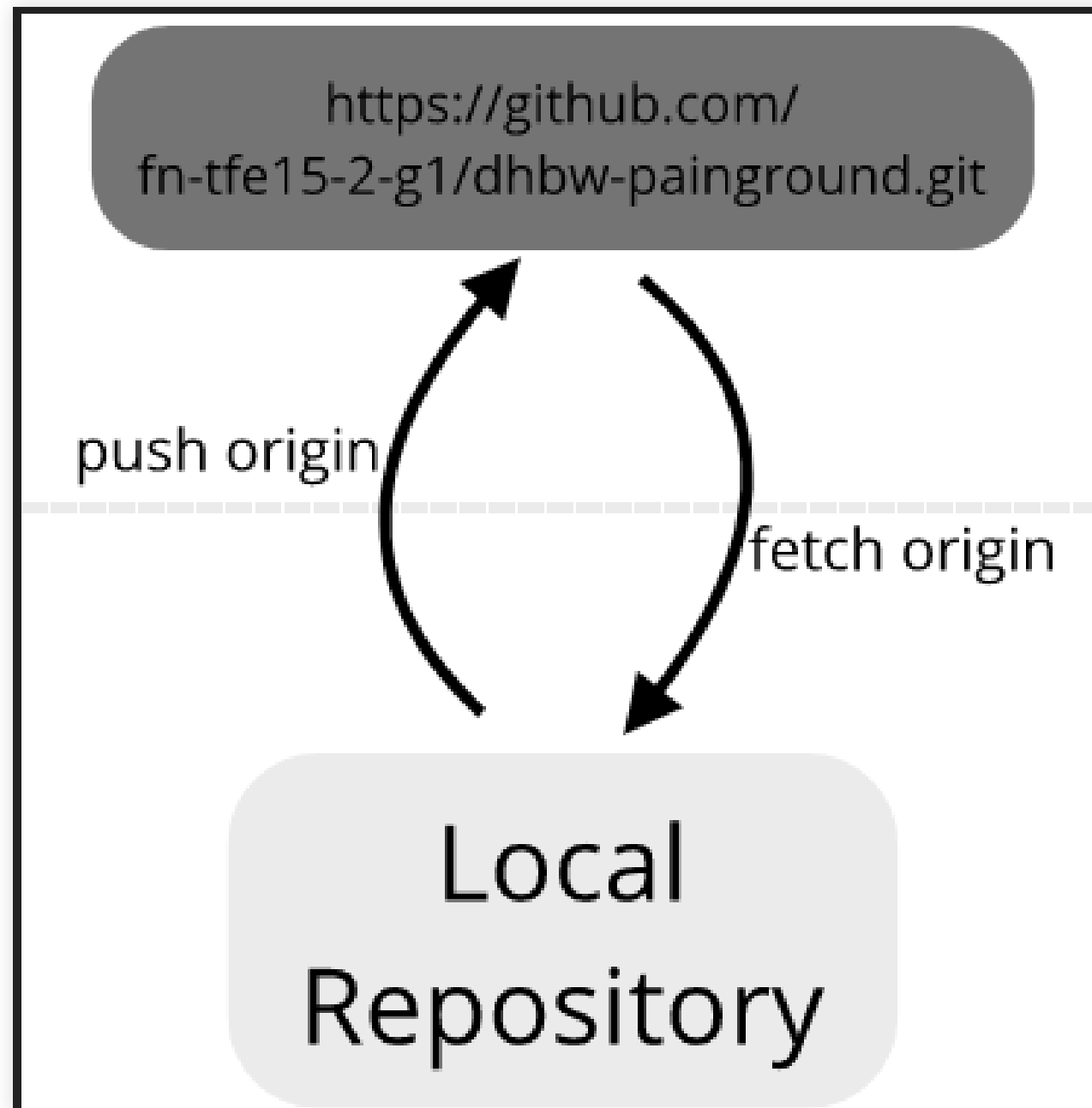
# CLONE & FORK

- `git clone`
  - Kopieren eines remote Repositories auf den eigenen Rechner
  - "erste Synchronisieren" plus "Checkout"
  - kein `git init` mehr nötig
- `fork`
  - kein Git Befehl
  - Findet auf einem Git-Server statt, z.B. auf <https://github.com>
  - im Hintergrund wird auch `git clone` ausgeführt

# CLONE & FORK

- Problem: Wie kommen Änderungen des Originals zu meinem Fork?
- Lösung: Original als weiteres Remote-Repo anlegen

# OHNE FORK



# OHNE FORK - ANDERS



# OHNE FORK - REMOTES

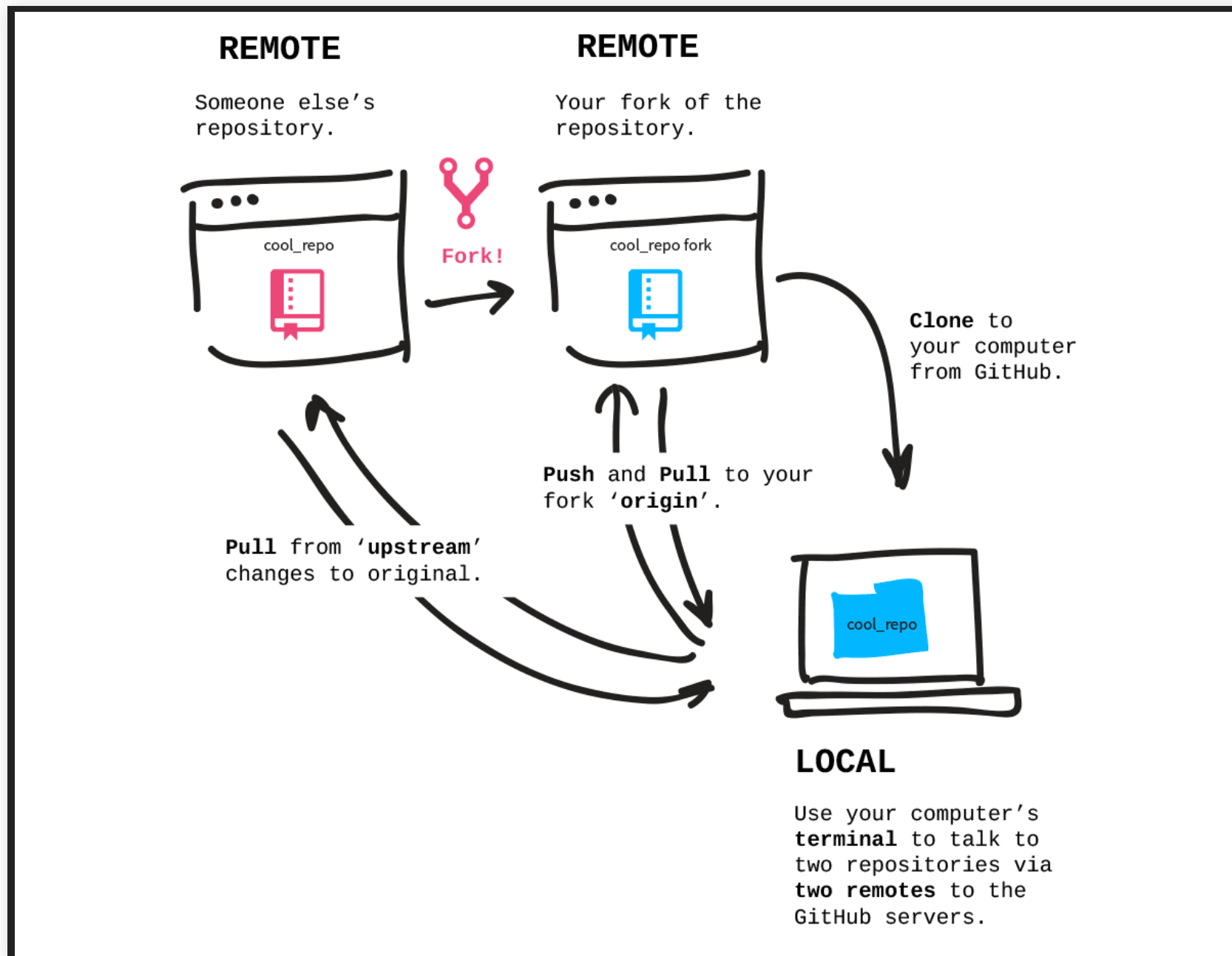
```
$ git remote -v  
origin    github.com/fn-tfe15-2-g1/dhbw-painground.git (fetch)  
origin    github.com/fn-tfe15-2-g1/dhbw-painground.git (push)
```

# MIT FORK





# MIT FORK - ANDERS



# MIT FORK - REMOTES

```
$ git remote add upstream https://github.com/barclay-reg/dhbw-paingro  
$ git remote -v  
origin      github.com/fn-tfe15-2-g1/dhbw-painground.git (fetch)  
origin      github.com/fn-tfe15-2-g1/dhbw-painground.git (push)  
upstream     github.com/barclay-reg/dhbw-painground.git (fetch)  
upstream     github.com/barclay-reg/dhbw-painground.git (push)
```

# MIT FORK - ÄNDERUNGEN ABHOLEN

```
## Änderungen von Remote "upstream" holen  
$ git fetch upstream  
## auf eigenen Branch "master" wechseln  
$ git checkout master  
## Alle commits von Branch "master" von Remote "upstream"  
## in aktuellen Branch mergen  
$ git merge upstream/master  
## Änderungen an github senden  
$ git push
```

# PULL REQUEST

- Antrag, ein oder mehrere Commits von einem Branch in einen anderen Branch zu mergen
  - sinnvoll von einem Fork zum Original
  - auch sinnvoll "innerhalb" eines einzigen Repos
- kann jemandem *zugewiesen* werden
- Erlaubt Code-Review, Code-Diskussion
- wenn Antrag akzeptiert ist, wird ein **Pull** (fetch & merge) gemacht
- Kann per `git request-pull` gestartet werden, aber
- besser per Web-Interface (Github, Bitbucket, Gitlab)

# WIESO PR

- Warum nicht einfach Mergen?
  - (Feature)-Branches bestehen manchmal länger
    - Niemand außer dem Author weiß, wann das Feature *fertig* ist
    - Erstellen des **PR** ist ein eindeutiger Trigger für
      - Start des Code-Reviews
      - Start von (langwierigen) automatisierten Tests

# WIESO PR

- **PR** im Umfeld von Original & Fork sind extrem hilfreich
  - Maintainer des Originals erlaubt nur wenigen das direkte Committen (und vor allem das Pushen) in das eigene Repo
  - mit Forks kann jeder Freiwillige trotzdem an dem Code arbeiten
  - mit dem Annehmen des **PR** erlaubt der Maintainer, die "fremden" Commits in sein Repo aufzunehmen