Git

16. Dezember 2021

Seminargruppe INF-Ba-12-21

Gliederung

Mathe-Prüfung

Nikolausklausur

... die erste Mathe-Klausur naht!

- **Termin:** `10.12.2020 16:40 Uhr`
- **Einschreibung:** endet am `7.12.2020`
- Hausaufgaben bringen Bonuspunkte!
- zeitig (am besten *vorgestern*) mit Lernen anfangen
- lernt zusammen!
- **Hilfsmittel:** ein A4-Blatt ein- oder beidseitig handschriftlich beschrieben

Eine 2.0 jetzt sichert euch das Modul!

Git

Was ist git?

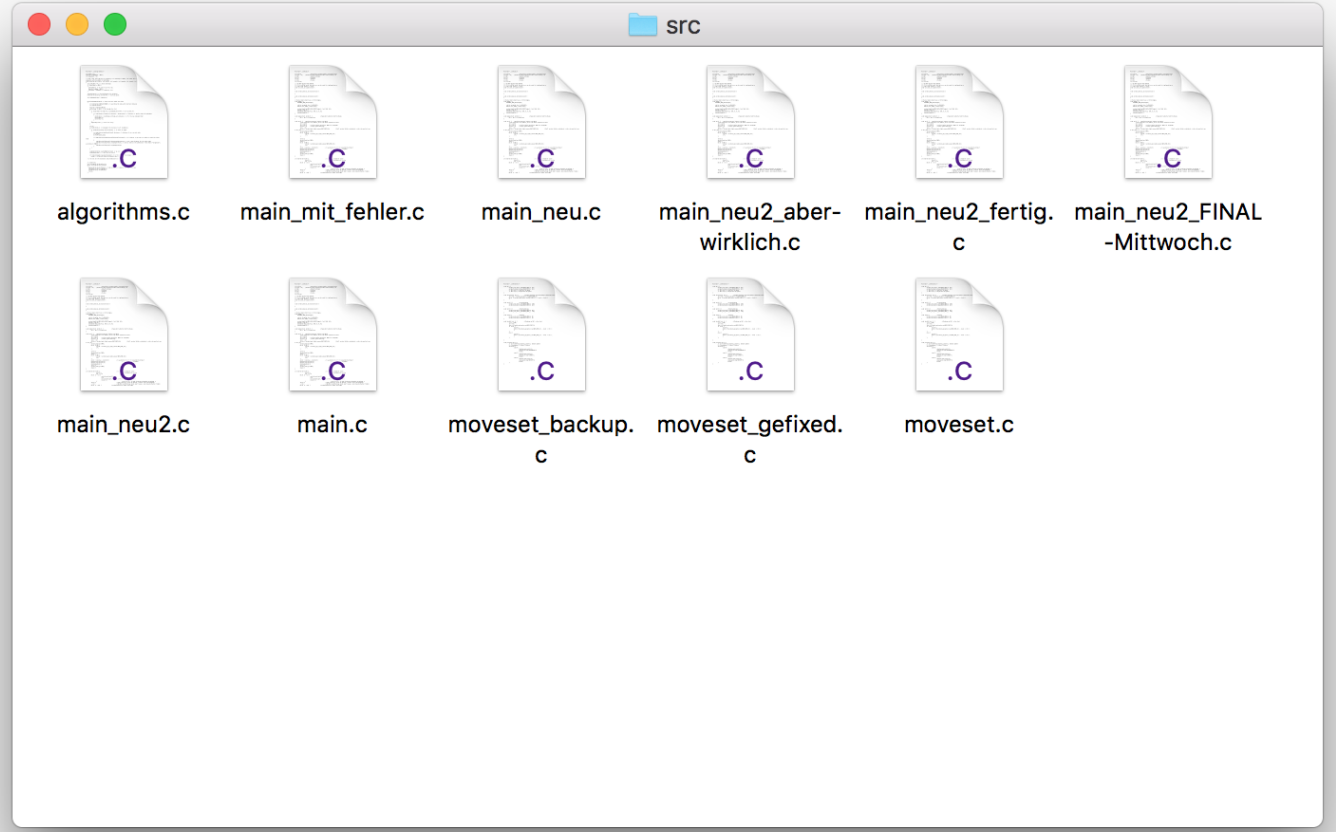
- Ein Versionsverwaltungssystem!

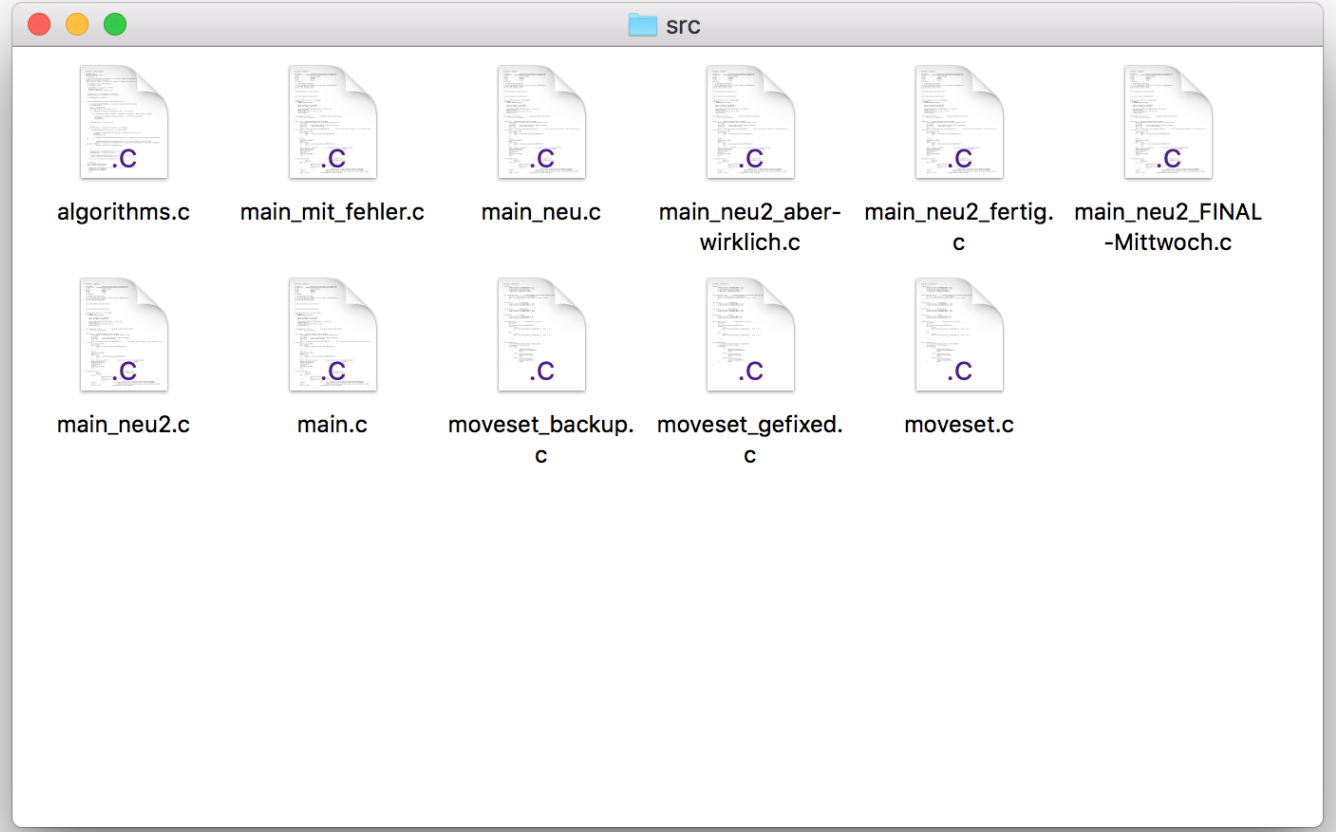
Und wozu ist das gut?

- Um den Überblick über verschiedene Versionen deines Projektes zu behalten
- Zum einfacheren Zusammenarbeiten mehrerer Personen an einem gemeinsamen Projekt
  - Deshalb im Robolab und SWP genutzt

Warum sollte ich git nutzen?

Was spricht gegen Dropbox? Oder Facebook?


src
algorithms.c
main_mit_fehler.c
main_neu.c
main_neu2_aber-wirklich.c
main_neu2_fertig.c
main_neu2_FINAL-Mittwoch.c
main_neu2.c
main.c
moveset_backup.c
moveset_gefixed.c
moveset.c


src
algorithms.c
main_mit_fehler.c
main_neu.c
main_neu2_aber-wirklich.c
main_neu2_fertig.c
main_neu2_FINAL-Mittwoch.c
main_neu2.c
main.c
moveset_backup.c
moveset_gefixed.c
moveset.c

Das.

Warum Versionierung?

- Dropbox ist (für solche Zwecke) unbenutzbar
  - man überschreibt gegenseitig Änderungen oder hat 20 Dateien à la

    `(In Konflikt stehende Version von xx am dd.mm.yyyy)`
- Verlust des Überblicks bei 20 Kopien der selben Datei mit anderem Namen

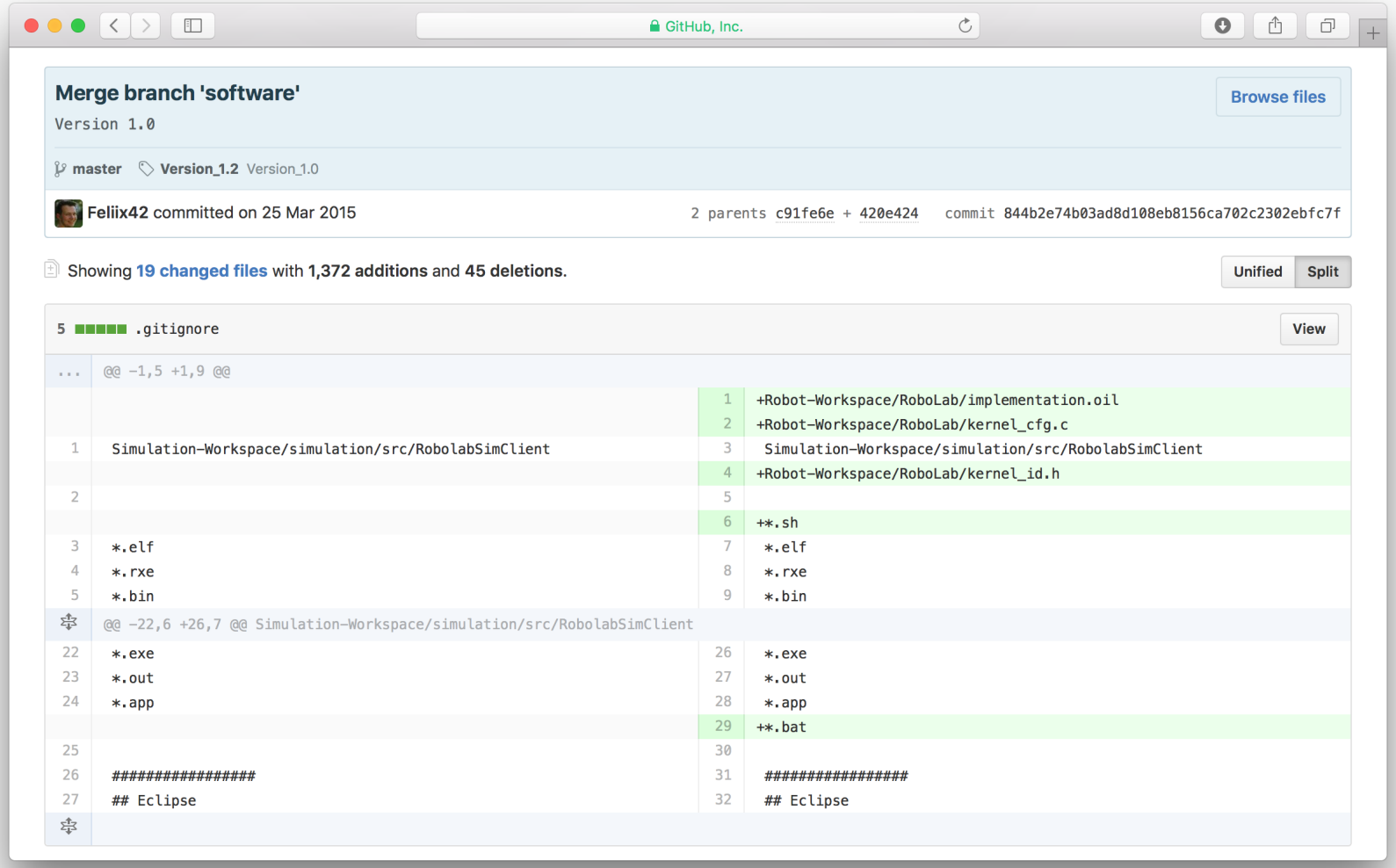
Warum Versionierung?

- Dropbox ist (für solche Zwecke) unbenutzbar
  - man überschreibt gegenseitig Änderungen oder hat 20 Dateien à la `(In Konflikt stehende Version von xx am dd.mm.yyyy)`
- Verlust des Überblicks bei 20 Kopien der selben Datei mit anderem Namen

Lösung:

- immer selbe Datei bearbeiten
- Zwischenstand irgendwo speichern, mit Info, was geändert wurde (zum rückgängig machen)

Wie das aussehen könnte...



grün: hinzugefügte Zeilen

weitere Vorteile

- man sieht, wer Fehler ins Programm eingeschleust hat
- zurück springen auf jeden beliebigen vorherigen Stand
  $\rightarrow$ jede Änderung *einzeln* rückgängig machbar
- auch online speicherbar

Programme zur Versionsverwaltung

Bekannte Programme:

- SVN
- Mercurial
- Git

Programme zur Versionsverwaltung

Bekannte Programme:

- SVN
- Mercurial
- Git - **zurzeit das Beliebteste**


git

Installation

Installation - Linux

Ubuntu/Debian

```
$ sudo apt-get install git
```

Arch Linux

```
$ sudo pacman -Sy git
```

Fedora

```
$ dnf -y install git
```

Installation - macOS

Über Homebrew (empfohlen)

```
$ brew install git
```

Download von der Webseite

https://git-scm.com

Installation - Windows

Installation des Linux Subsystems

Danach, wie bei der gewählten Linuxdistibution

Clients

Es gibt eine Reihe von grafischen Clients, die auf dem Kommandozeilenprogramm aufsetzen. Empfehlenswert sind:

- GitKraken
- SourceTree (viele Konfigurationsmöglichkeiten)
- GitHub Desktop
- IDE

Aaaaber: Lieber Kommandozeilen-Client nutzen, um die Funktionsweise zu lernen.

Hands on!

Zeit für eine praktische Einführung!

Vorbereitung

- ☐ `git` installiert?
- ☐ neuen (leeren) Ordner anlegen
- ☐ Linux Subsystem (Windows) oder Terminal öffnen
  per `cd pfad/zum/ordner` in den neuen Ordner wechseln

Navigation in einem Terminal

noch nie eine Shell benutzt?

Navigation in einem Terminal

noch nie eine Shell benutzt?

- `cd mein_ordner` wechsle in `mein_ordner`
- `mkdir mein_ordner` erstelle `mein_ordner`
- `ls` liste mir alle Datein/Ordner im aktuellen Verzeichnis
- `touch meine_datei` erstelle `meine_datei`
- `cat meine_datei` zeige den Inhalt von `meine_datei` an
- `rm meine_datei` lösche `meine_datei`

Initiales Setup

Noch nie `git` benutzt?

Initiales Setup

Noch nie `git` benutzt?

```
git config --global user.name "[name]"
```

Setzt Namen, der unter dem Commit stehen wird.

Initiales Setup

Noch nie `git` benutzt?

```
git config --global user.name "[name]"
```

Setzt Namen, der unter dem Commit stehen wird.

```
git config --global user.email "[email]"
```

Mailadresse des Users (**wichtig:** sollte GitHub-Adresse sein!)

Initiales Setup

Noch nie `git` benutzt?

`git config --global user.name "[name]"`

Setzt Namen, der unter dem Commit stehen wird.

`git config --global user.email "[email]"`

Mailadresse des Users (**wichtig:** sollte GitHub-Adresse sein!)

```
[felix@Samaritan] $ git config --global user.name "Max Mustermann"
[felix@Samaritan] $ git config --global user.email "mustermann@ifsr.de"
```

Und am Anfang war...

```
git init
```

Erstellt ein neues Repository *(Lager)* im aktuellen Ordner.

Und am Anfang war...

`git init`

Erstellt ein neues Repository *(Lager)* im aktuellen Ordner.

```
[felix@Samaritan] $ git init
Initialized empty Git repository in /Users/felix/test/.git/
```

Und was ist jetzt passiert?

- `.git`-Ordner wurde angelegt
- in ihm geschieht „Magie“ hinter den Kulissen
- dient als Ablage für gespeicherte Änderungen an Dateien
- Einstellungen für Repository liegen hier

In diesem Ordner müsst ihr für gewöhnlich nichts ändern.

Ein kleiner Blick unter die Haube...

`git` - Workflow

Intern organisiert `git` die überwachten Dateien in 3 Bereichen:

Working Directory Da liegen alle bearbeiteten Dateien aus eurem Projekt-Ordner drin.

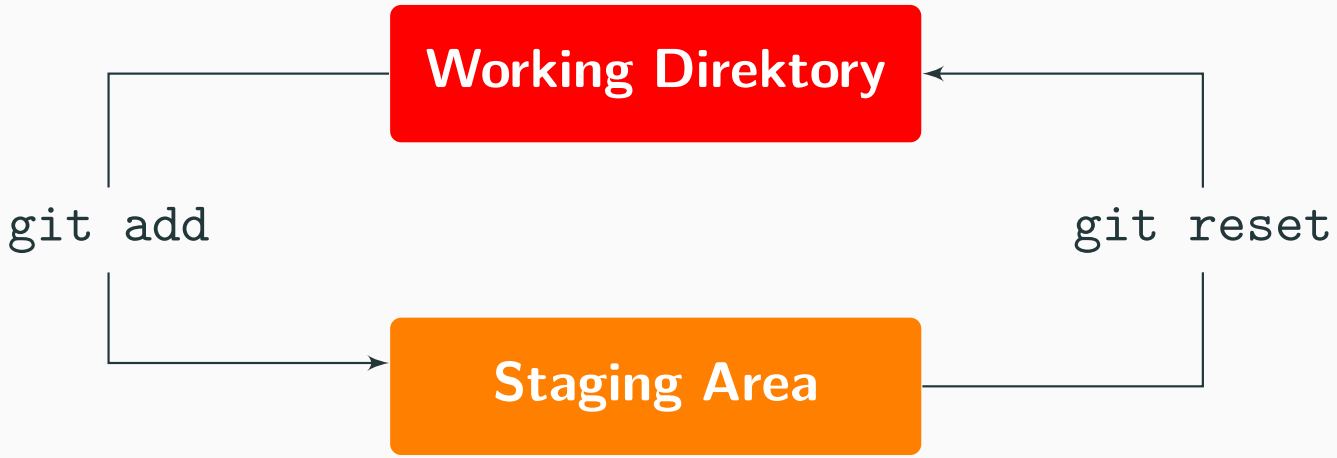
Staging Area Wollt ihr euren Zwischenstand speichern, könnt ihr einzelne Änderungen (einzelne Dateien oder sogar Zeilen) dafür auswählen. Diese landen dann hier.

Repository Nachdem die ausgewählten Änderungen aus der Staging Area committed wurden, liegen sie als Gesamtpaket (Commit) hier.

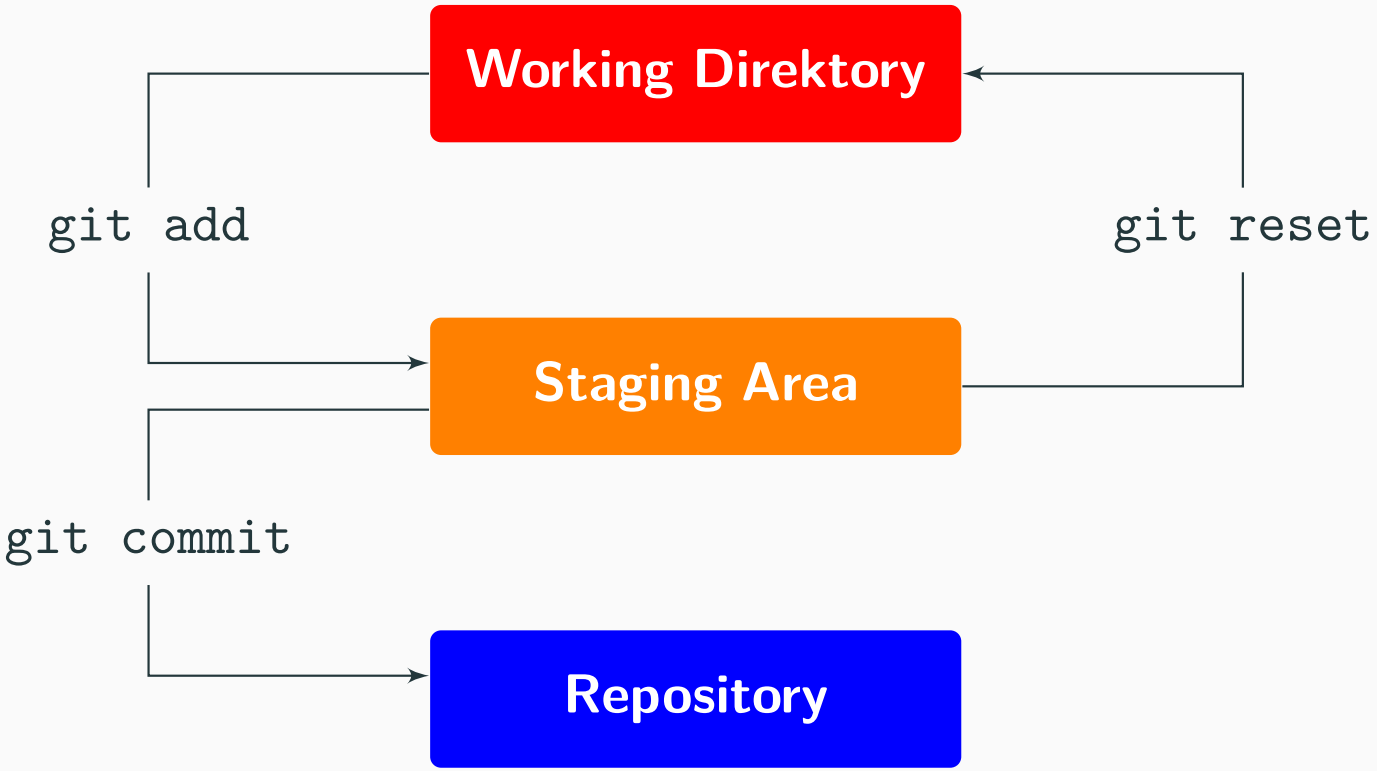
`git` - Workflow

Working Direktory

`git` - Workflow



`git` - Workflow



Zurück zur Praxis!

Her mit dem Inhalt!

(Normalerweise kommt jetzt der Teil, an dem ihr programmiert.)

Erstellt eine Textdatei und schreibt ein paar Zeilen.

Änderungen speichern

Genug Inhalt zusammengekommen?
Zeit für den *Commit*.

Änderungen speichern

Genug Inhalt zusammengekommen?
Zeit für den *Commit*.

Den was...?

Commitment leicht gemacht

Ein Commit ist der Vorgang, bei dem die geänderten Dateien aus dem Working Tree gespeichert werden.

Das spielt sich nach dem immer gleichen Schema ab:

1. Dateien bearbeiten
2. zu speichernde Änderungen auswählen (addn oder stagen)
3. Änderungen *committen*

Änderungen speichern (1)

Mit `git status` wird Status des Repositories angezeigt:

```
[felix@Samaritan] $ git status
On branch main

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    hallowelt.txt

nothing added to commit but untracked files present (use "git
    add" to track)
```

Änderungen speichern (2)

Jetzt müssen alle geänderten Dateien *gestaged* werden:

```
git add [Datei(en)]
```

```
[felix@Samaritan] $ git add hallowelt.txt
```

Änderungen speichern (2)

Jetzt müssen alle geänderten Dateien *gestaged* werden:

`git add [Datei(en)]`

```
[felix@Samaritan] $ git add hallowelt.txt
```

Immer überlegen, *welche* Dateien geaddet werden sollen. Passwörter und Access-Token haben in git nichts zu suchen!

Wo stehen wir jetzt?

Probiert doch noch mal `git status`...

Wo stehen wir jetzt?

```
[felix@Samaritan] $ git status
On branch main

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   hallowelt.txt
```

Zeit, den Sack zu zu machen!

Bleibt nur noch der eigentliche *Commit* übrig!

```
git commit -m "[Nachricht]"
```

```
[felix@Samaritan] $ git commit -m "Add hallowelt.txt"

[main (root-commit) 6386d2f] Add hallowelt.txt
 1 file changed, 1 insertion(+)
 create mode 100644 hallowelt.txt
```

Glückwunsch! Euer erster Commit! :)

Tipp: Verwendet *aussagekräftige* Commit-Nachrichten, aus denen man sofort herauslesen kann, welche Änderungen ihr vorgenommen habt!

Negativ-Beispiel:

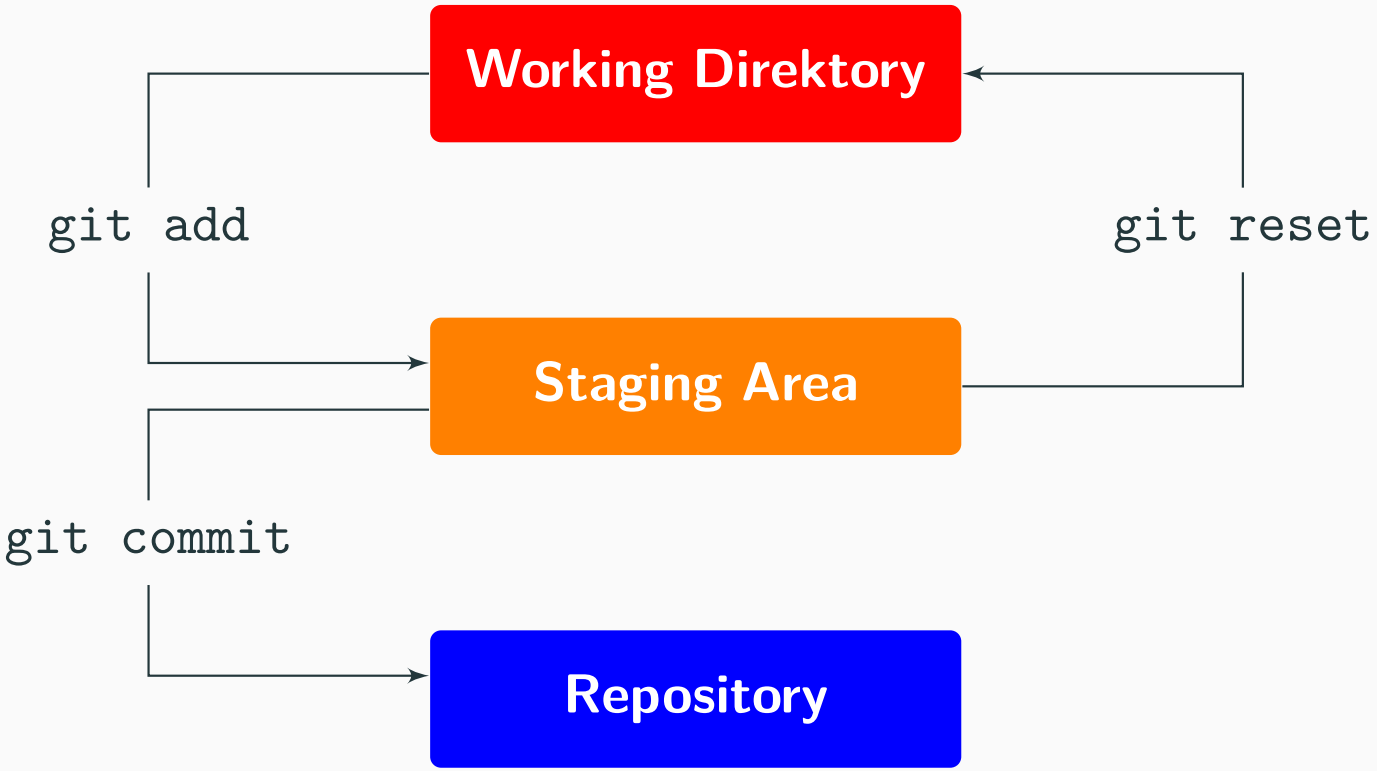
```
3d06ff9 Aariesntdyaus
9bffefc Fuuuuuuuuuuu
1829923 I are the master of code!!!111
3d06ff9 Added/removed a file
9bffefc This time it is probably fixed
1829923 Prettify the output
```

Nett. Aber alles immer noch lokal. Wie funktioniert jetzt die Zusammenarbeit?

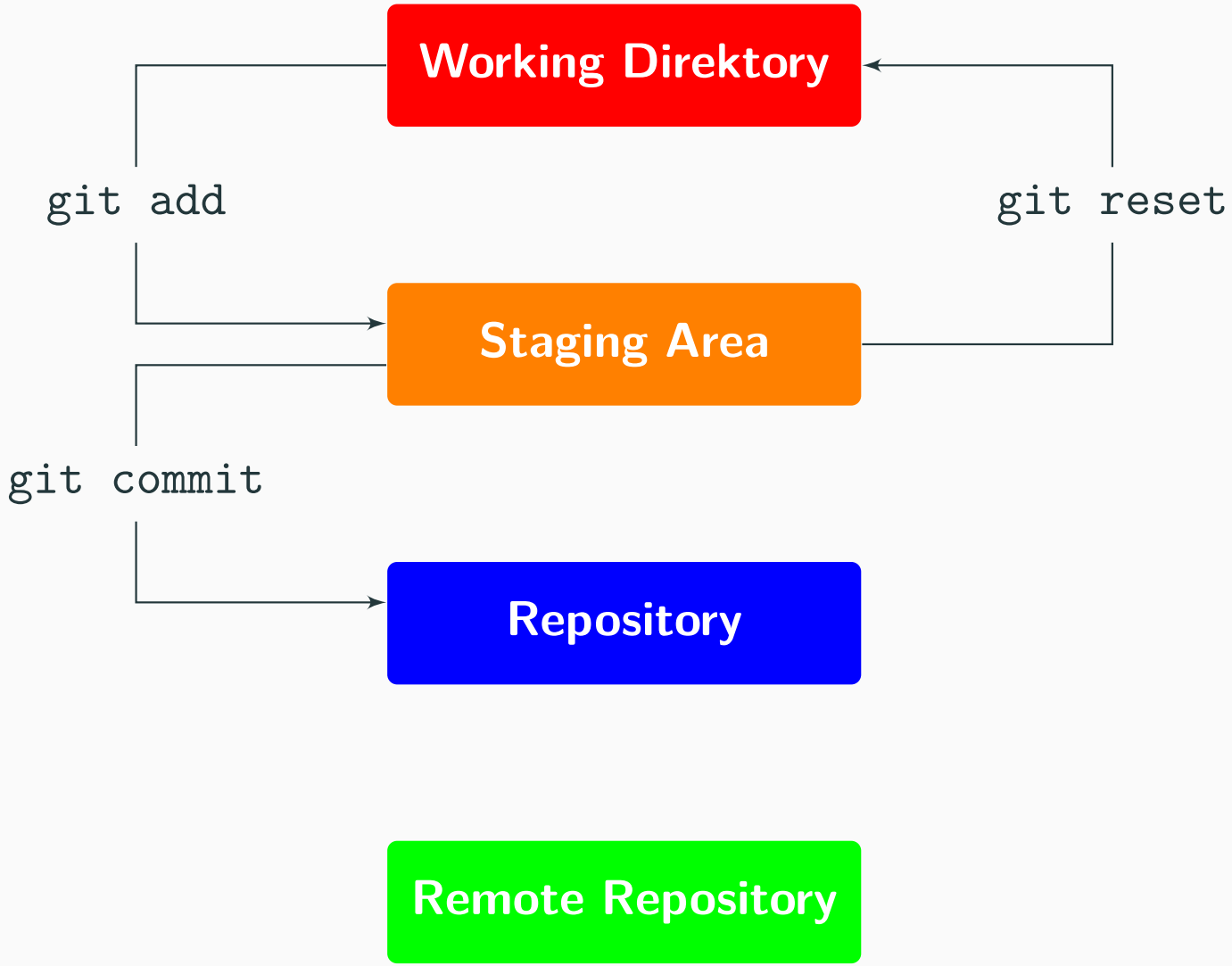
Nett. Aber alles immer noch lokal. Wie funktioniert jetzt die Zusammenarbeit?

Mit Remote Repositories!

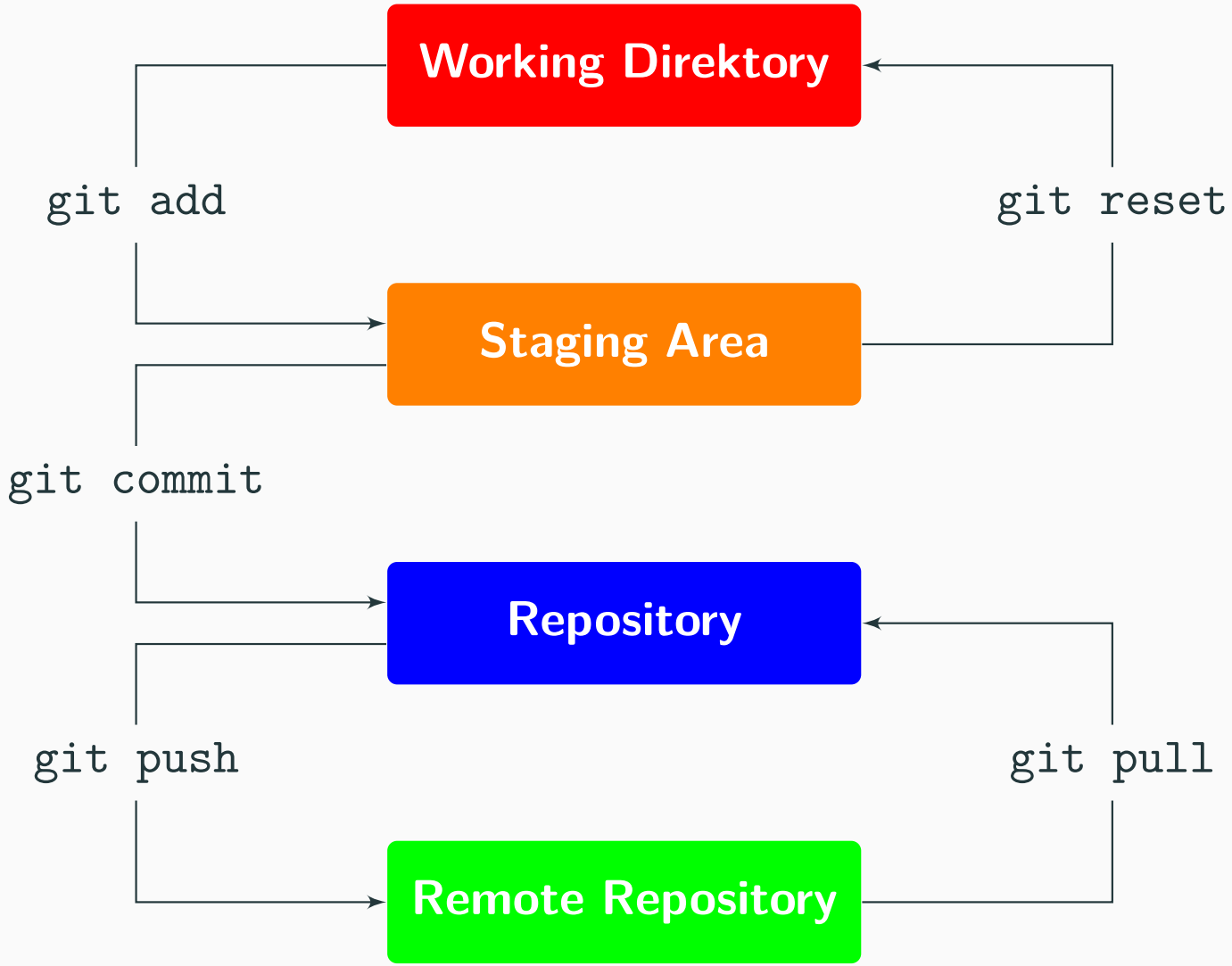
`git` - Workflow



`git` - Workflow



`git` - Workflow



Ab in die Cloud! - Befehle

`git push [Remote] [Branch]`

lädt eure Commits hoch (man nennt es auch *pushen*)

`git pull [Remote] [Branch]`

lädt fremde Änderungen runter (man nennt es auch *pullen*)

Ab in die Cloud! - Befehle

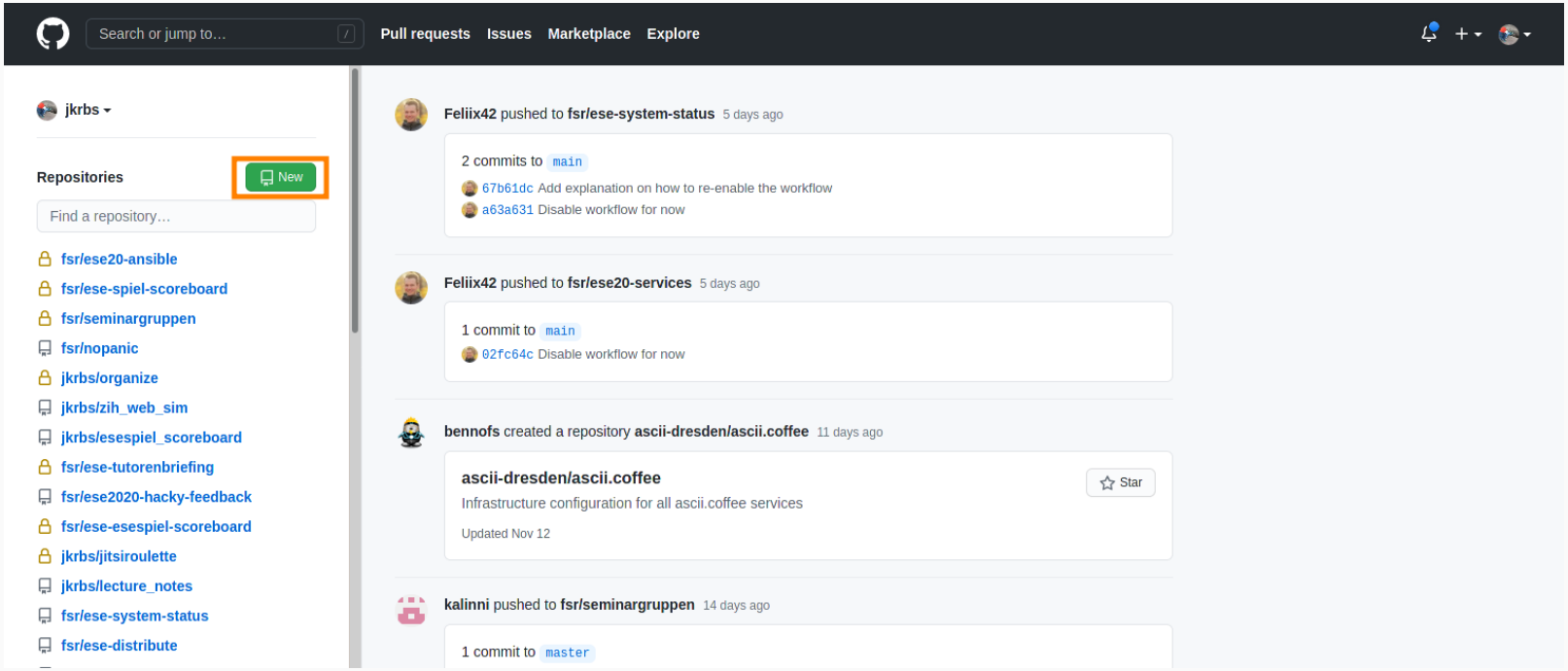
`git push [Remote] [Branch]`

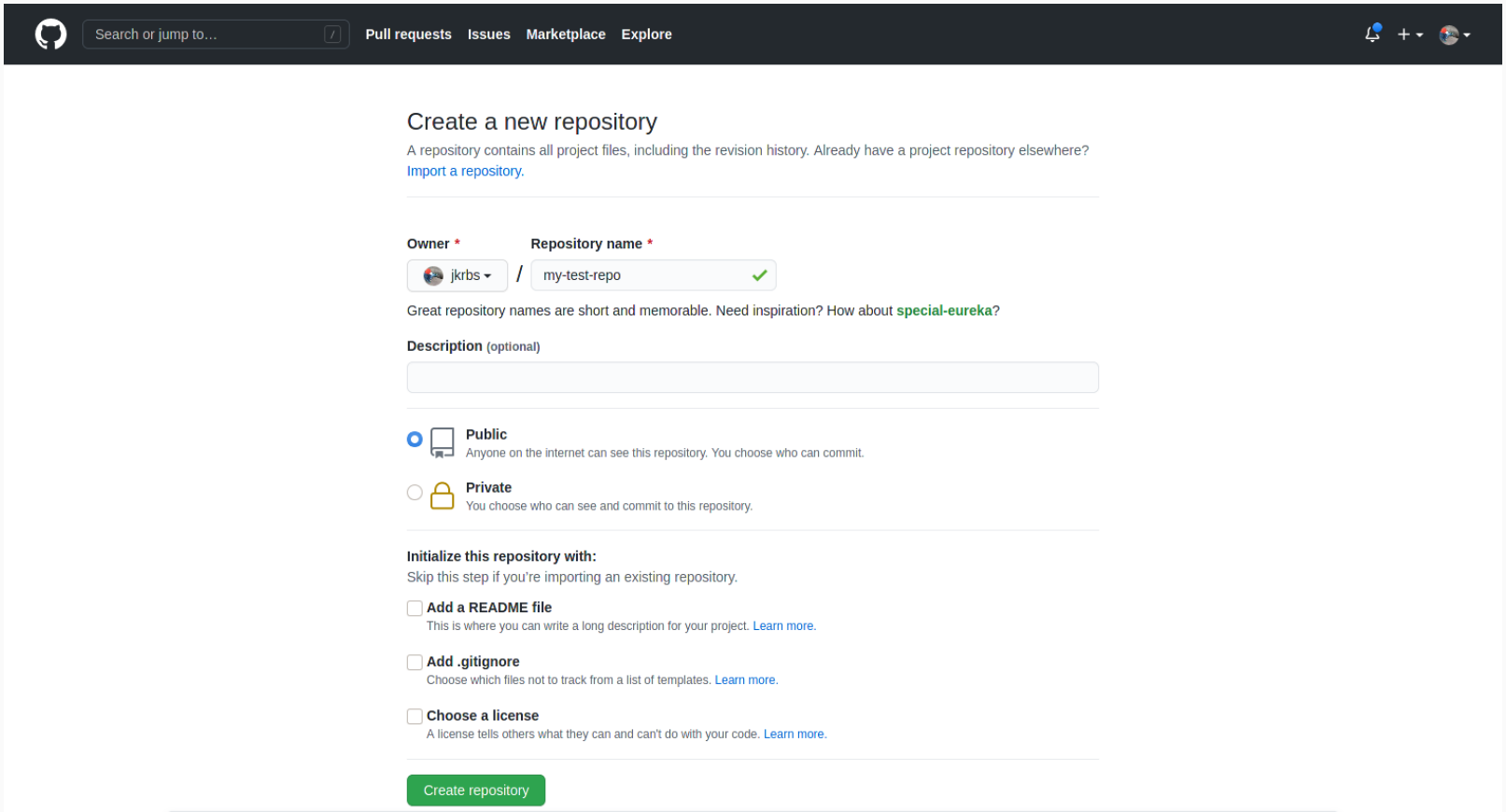
lädt eure Commits hoch (man nennt es auch *pushen*)

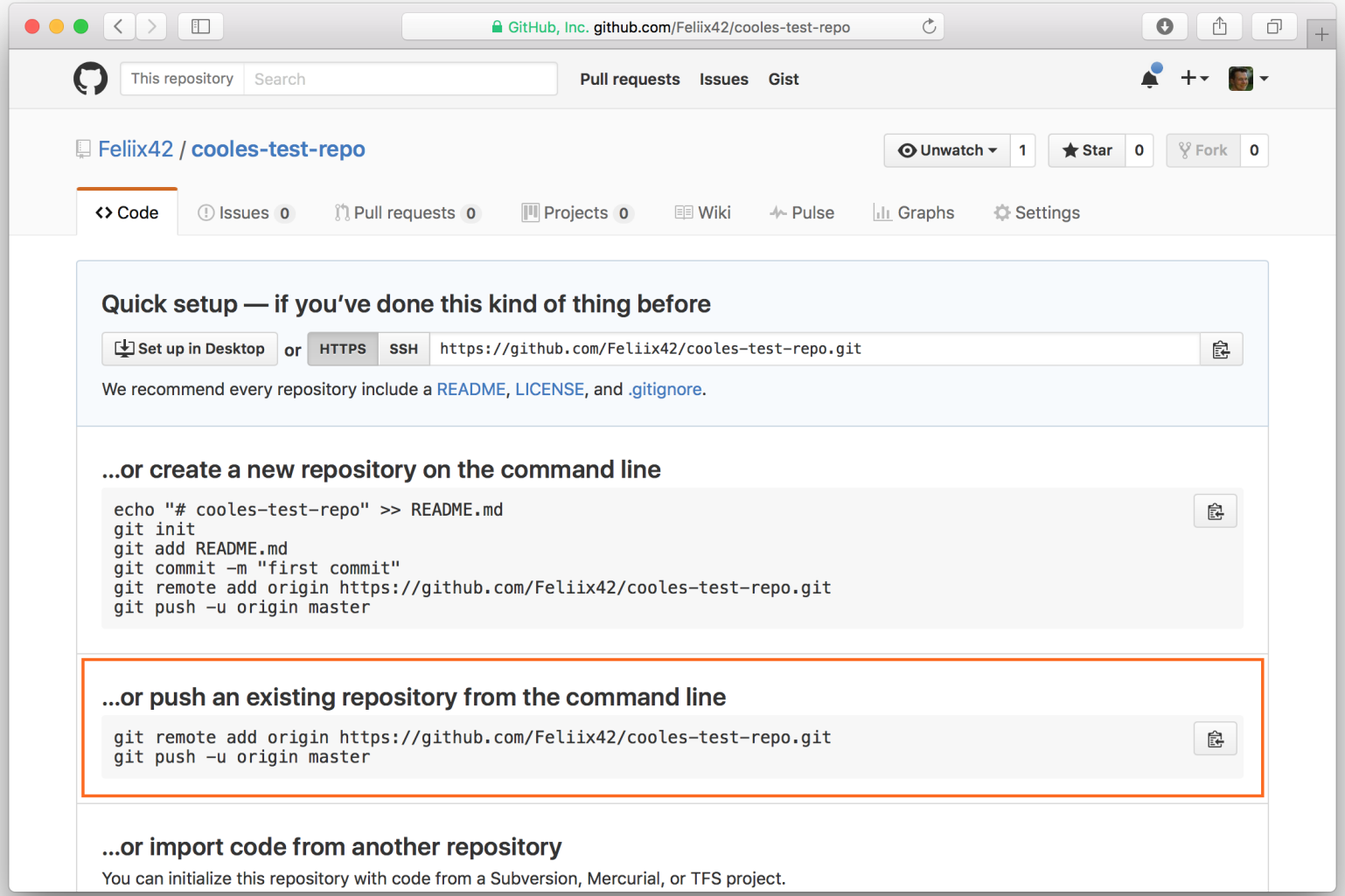
`git pull [Remote] [Branch]`

lädt fremde Änderungen runter (man nennt es auch *pullen*)

Aber dazu brauchen wir erstmal ein Remote Repository...

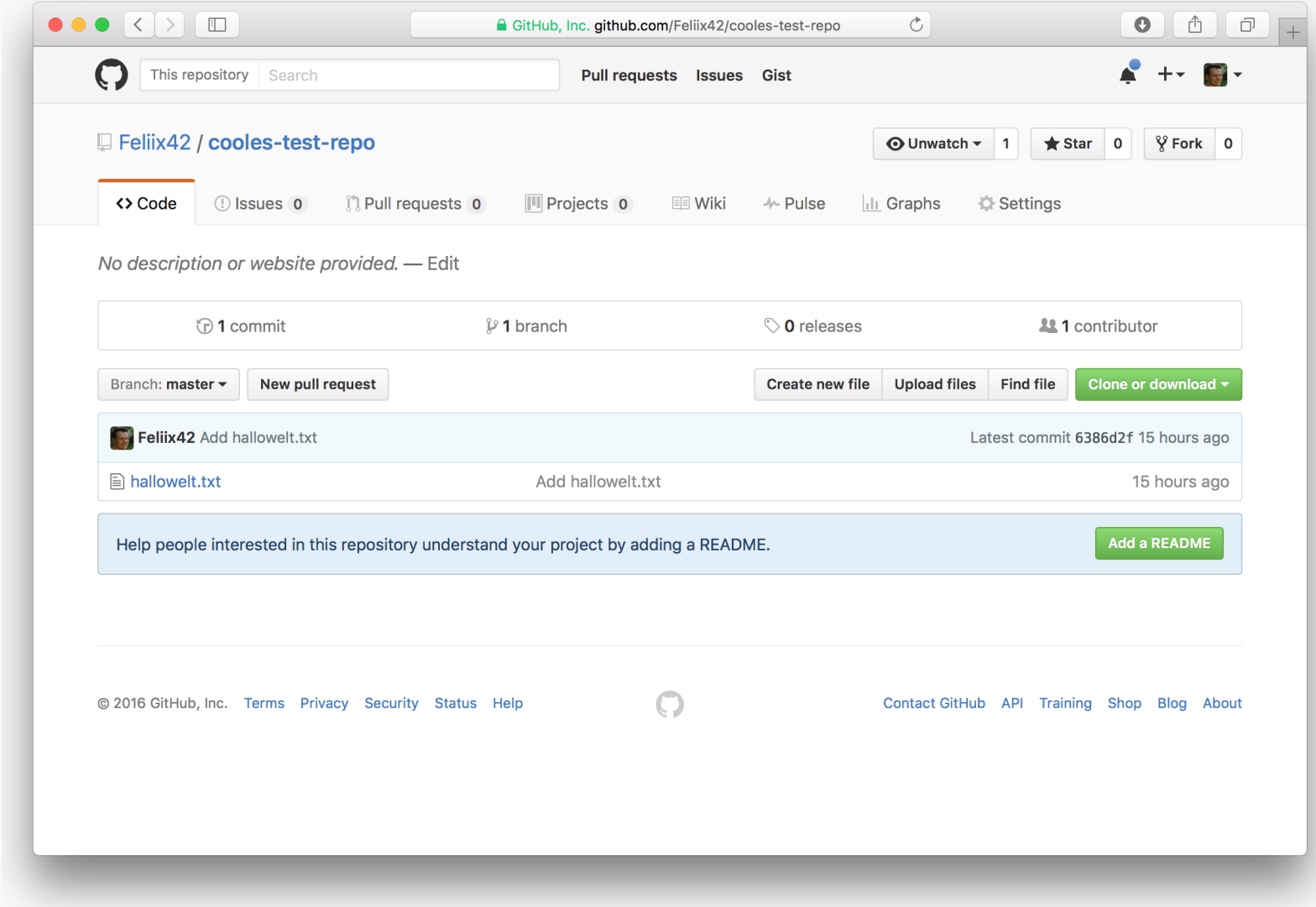

Search or jump to…
Pull requests
Issues
Marketplace
Explore
jkrbs
Repositories
New
Find a repository…
fsr/ese20-ansible
fsr/ese-spiel-scoreboard
fsr/seminargruppen
fsr/nopanic
jkrbs/organize
jkrbs/zih_web_sim
jkrbs/esespiel_scoreboard
fsr/ese-tutorenbriefing
fsr/ese2020-hacky-feedback
fsr/ese-esespiel-scoreboard
jkrbs/jitsiroulette
jkrbs/lecture_notes
fsr/ese-system-status
fsr/ese-distribute
Feliix42 pushed to fsr/ese-system-status 5 days ago
2 commits to main
67b61dc Add explanation on how to re-enable the workflow
a63a631 Disable workflow for now
Feliix42 pushed to fsr/ese20-services 5 days ago
1 commit to main
02fc64c Disable workflow for now
bennofs created a repository ascii-dresden/ascii.coffee 11 days ago
ascii-dresden/ascii.coffee
Infrastructure configuration for all ascii.coffee services
Updated Nov 12
Star
kalinni pushed to fsr/seminargruppen 14 days ago
1 commit to master


Search or jump to…
Pull requests
Issues
Marketplace
Explore
Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.
Owner *
Repository name *
jkrbs
/
my-test-repo
Great repository names are short and memorable. Need inspiration? How about special-eureka?
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with:
Skip this step if you're importing an existing repository.
Add a README file
This is where you can write a long description for your project. Learn more.
Add .gitignore
Choose which files not to track from a list of templates. Learn more.
Choose a license
A license tells others what they can and can't do with your code. Learn more.
Create repository


GitHub, Inc. github.com/Feliix42/cooles-test-repo
This repository
Search
Pull requests
Issues
Gist
Feliix42 / cooles-test-repo
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
Settings
Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/Feliix42/cooles-test-repo.git
We recommend every repository include a README, LICENSE, and .gitignore.
...or create a new repository on the command line
echo "# cooles-test-repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/Feliix42/cooles-test-repo.git
git push -u origin master
...or push an existing repository from the command line
git remote add origin https://github.com/Feliix42/cooles-test-repo.git
git push -u origin master
...or import code from another repository
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Ab in die Cloud! (2)

```
[felix@Samaritan] $ git remote add origin https://github.com/
    Feliix42/cooles-test-repo.git
[felix@Samaritan] $ git push -u origin main
Username for 'https://github.com': Feliix42
Password for 'https://Feliix42@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 869 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Feliix42/cooles-test-repo.git
 * [new branch]      main -> main
Branch main set up to track remote branch main from origin.
```



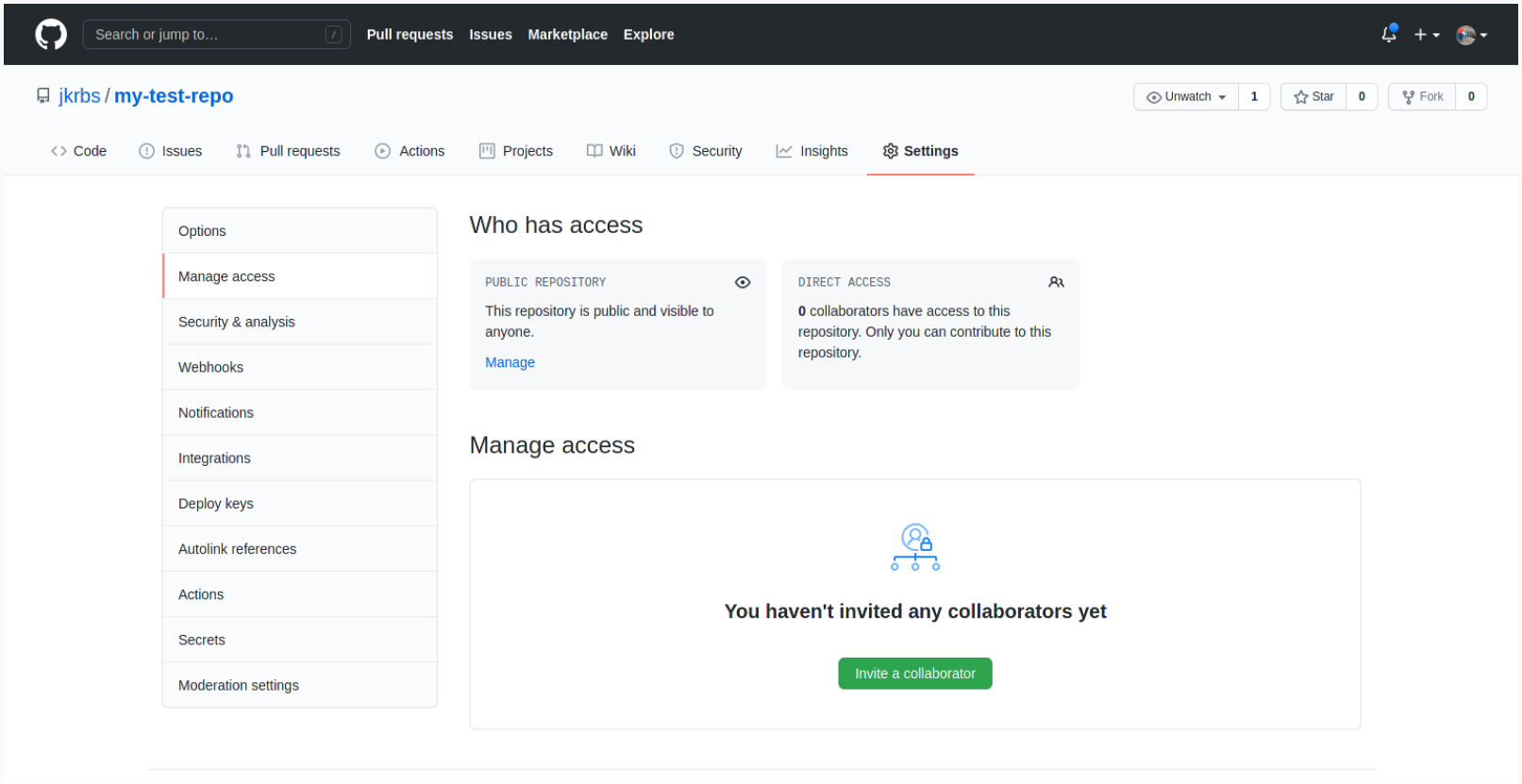
Es funktioniert!

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu

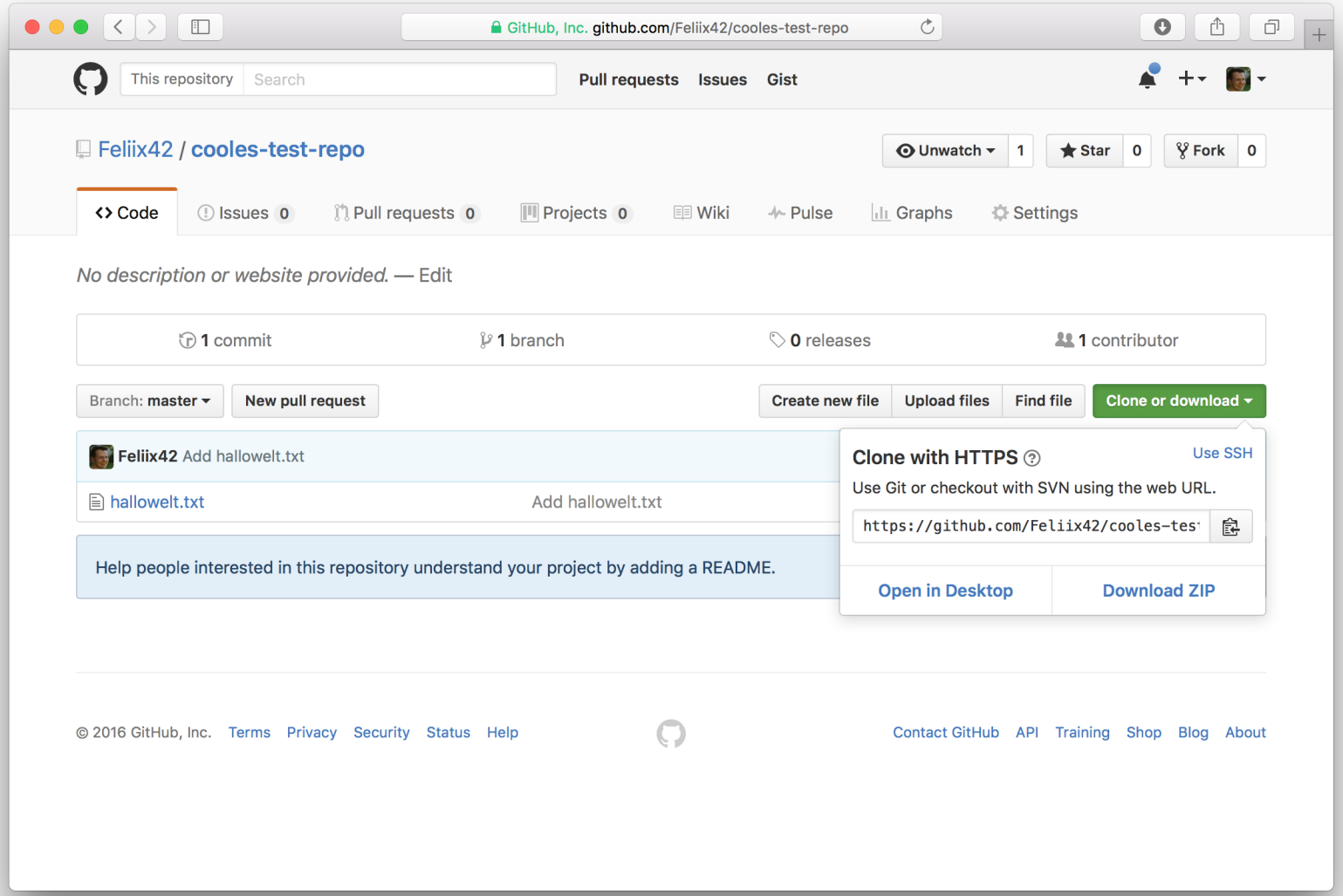

Search or jump to…
Pull requests
Issues
Marketplace
Explore
jkrbs / my-test-repo
Unwatch 1
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Options
Manage access
Security & analysis
Webhooks
Notifications
Integrations
Deploy keys
Autolink references
Actions
Secrets
Moderation settings
Who has access
PUBLIC REPOSITORY
This repository is public and visible to anyone.
Manage
DIRECT ACCESS
0 collaborators have access to this repository. Only you can contribute to this repository.
Manage access
You haven't invited any collaborators yet
Invite a collaborator

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu
- Partner 2 klont das Repository von Partner 1


GitHub, Inc. github.com/Feliix42/cooles-test-repo
This repository
Search
Pull requests
Issues
Gist
Feliix42 / cooles-test-repo
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
Settings
No description or website provided. — Edit
1 commit
1 branch
0 releases
1 contributor
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
Feliix42 Add hallowelt.txt
hallowelt.txt
Add hallowelt.txt
Help people interested in this repository understand your project by adding a README.
Clone with HTTPS
Use SSH
Use Git or checkout with SVN using the web URL.
https://github.com/Feliix42/cooles-tes
Open in Desktop
Download ZIP
© 2016 GitHub, Inc.
Terms
Privacy
Security
Status
Help
Contact GitHub
API
Training
Shop
Blog
About

```
[harold@Machine] $ git clone https://github.com/Feliix42/cooles-
    test-repo.git
Cloning into 'cooles-test-repo'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu
- Partner 2 klont das Repository von Partner 1

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu
- Partner 2 klont das Repository von Partner 1
- Bearbeitet beide `hallowelt.txt`
  (fügt z.B. den Satz „Ich heiße [...]“ hinzu und löscht den Rest.)

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu
- Partner 2 klont das Repository von Partner 1
- Bearbeitet beide `hallowelt.txt`
  (fügt z.B. den Satz „Ich heiße [...]“ hinzu und löscht den Rest.)
- Einer von euch committed und pusht seine Änderungen
  (`git add [...]`, `git commit -m "[...]"`, `git push origin main`)

Zum Abschluss: Kollaboration!

- findet euch zu zweit zusammen
- Partner 1 fügt Partner 2 zu seinem Repository hinzu
- Partner 2 klont das Repository von Partner 1
- Bearbeitet beide `hallowelt.txt`
  (fügt z.B. den Satz „Ich heiße [...]“ hinzu und löscht den Rest.)
- Einer von euch committed und pusht seine Änderungen
  (`git add [...]`, `git commit -m "[...]"`, `git push origin main`)
- Dann macht der andere Partner das gleiche

```
[felix@Samaritan] $ git commit -m "Ich bin Felix"

[main 50126f1] Ich bin Felix
 1 file changed, 1 insertion(+), 1 deletion(-)

[felix@Samaritan] $ git push origin main
To https://github.com/Feliix42/cooles-test-repo.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/Feliix42/
    cooles-test-repo.git'
hint: Updates were rejected because the remote contains work
hint: that you do not have locally. This is usually caused
hint: by another repository pushing to the same ref. You may
hint: want to first integrate the remote changes (e.g.,
hint: 'git pull ...') before pushing again. See the 'Note
hint: about fast-forwards' in 'git push --help' for details.
```

Huch.

Euer `push` wurde zurückgewiesen, weil es neuere Änderungen gibt, die ihr noch nicht auf eurem PC habt.

Lösung:

```
git pull origin main
```

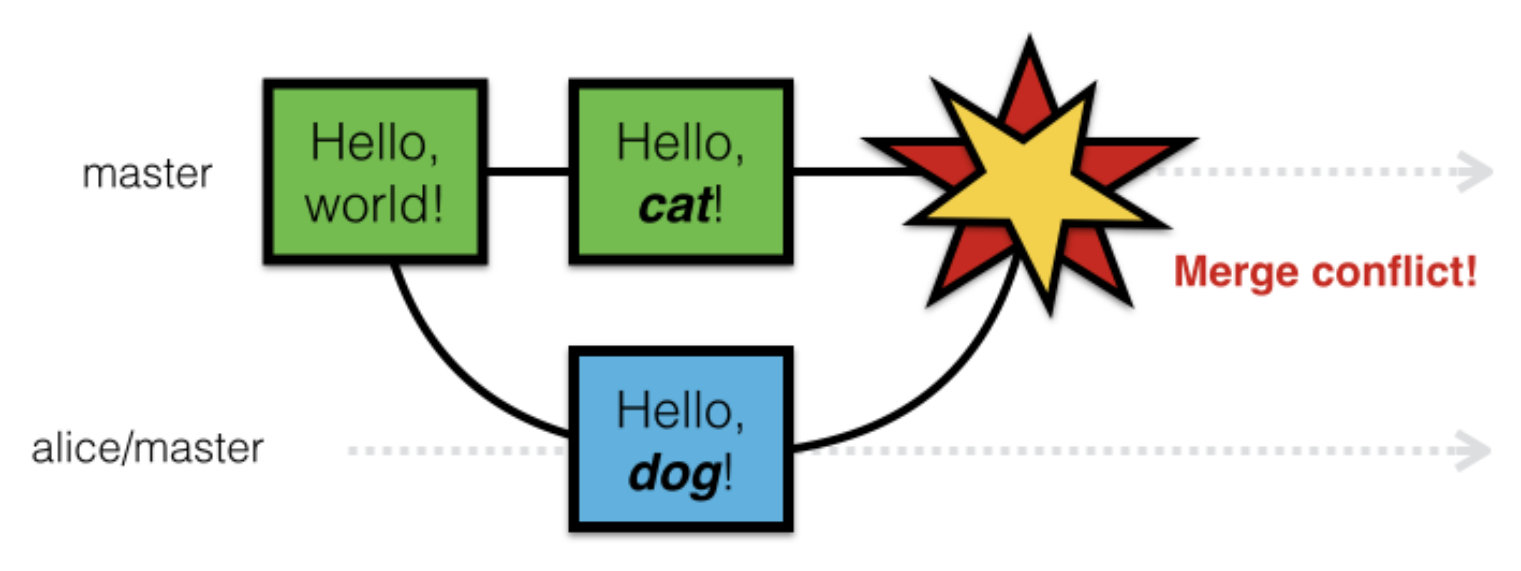
```
[felix@Samaritan] $ git pull origin main
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/Feliix42/cooles-test-repo
   6386d2f..90188bf  main     -> origin/main
Auto-merging hallowelt.txt
CONFLICT (content): Merge conflict in hallowelt.txt
Automatic merge failed; fix conflicts and then commit the result
```

```
[felix@Samaritan] $ git pull origin main
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/Feliix42/cooles-test-repo
   6386d2f..90188bf  main     -> origin/main
Auto-merging hallowelt.txt
CONFLICT (content): Merge conflict in hallowelt.txt
Automatic merge failed; fix conflicts and then commit the result
```

Glückwunsch! Euer erster *Merge-Conflict!*


GIT PULL
CONFLICT
memegenerator.net

Okay... Und jetzt?



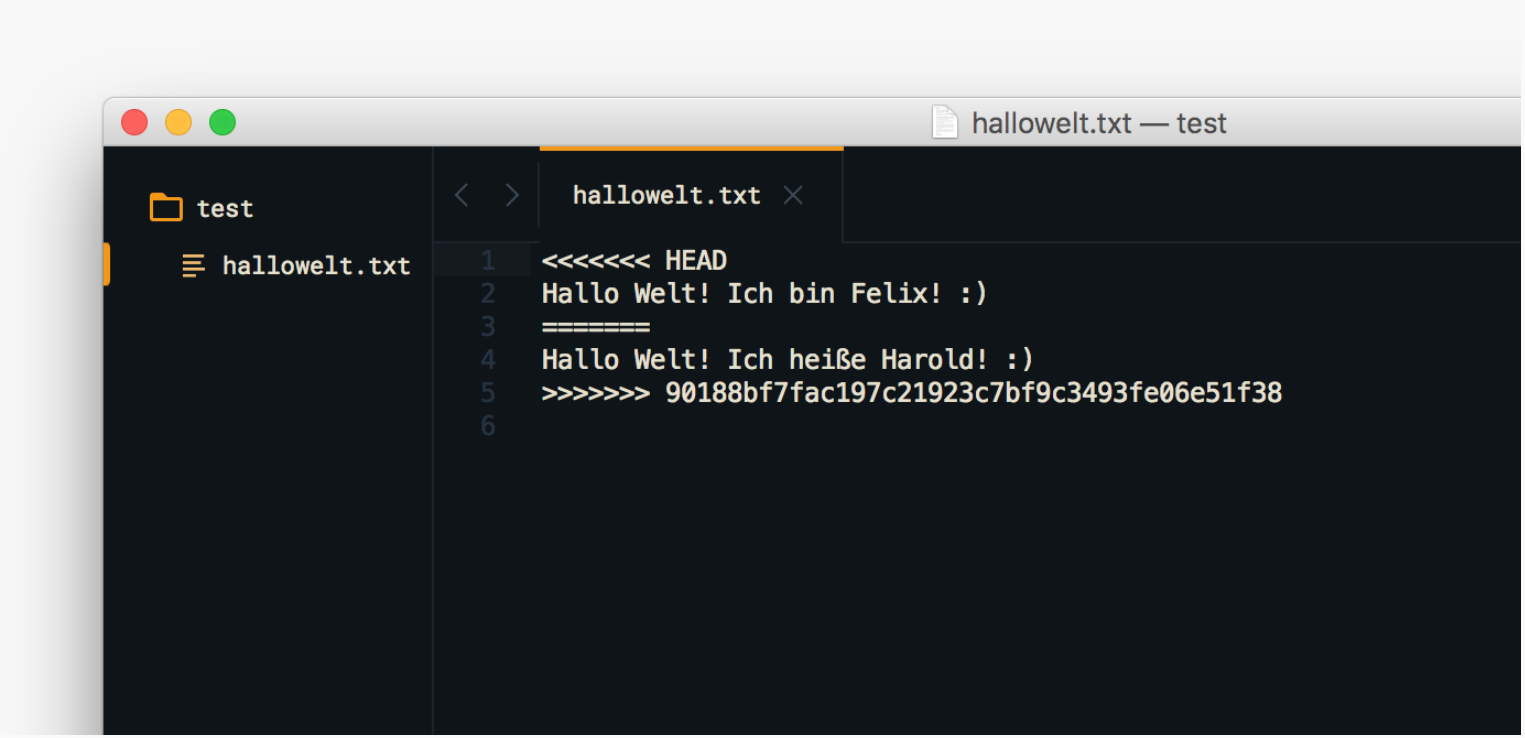
Überschneiden sich die Änderungen von zwei Commits, die `git` nicht automatisch beheben kann, entsteht ein **Merge Conflict**, der von Hand behoben werden muss.

Wie?

Öffnet `hallowelt.txt` im Texteditor.
Das sollte etwa so aussehen:

Wie?

Öffnet `hallowelt.txt` im Texteditor.
Das sollte etwa so aussehen:



Was sehe ich da?

- oberhalb des ======= stehen eure Änderungen (markiert mit *HEAD*)
- darunter die Änderung des Konflikte verursachenden Commits (markiert mit der *Commit-ID*)

Was sehe ich da?

- oberhalb des ======= stehen eure Änderungen (markiert mit *HEAD*)
- darunter die Änderung des Konflikte verursachenden Commits (markiert mit der *Commit-ID*)

- kompletter Abschnitt zwischen <<<< und >>>> muss jetzt von Hand gemerged werden
- löscht die Änderungen, die weg sollen und fasst alles so zusammen, dass es euch passt

Was sehe ich da?

- oberhalb des `=======` stehen eure Änderungen (markiert mit *HEAD*)
- darunter die Änderung des Konflikte verursachenden Commits (markiert mit der *Commit-ID*)

- kompletter Abschnitt zwischen `<<<<` und `>>>>` muss jetzt von Hand gemerged werden
- löscht die Änderungen, die weg sollen und fasst alles so zusammen, dass es euch passt
- seht euch mit `git status` den Zwischenstand an

```
[felix@Samaritan] $ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   hallowelt.txt

no changes added to commit (use "git add" and/or "git commit -a
    ")
```

Was sehe ich da?

- oberhalb des `=======` stehen eure Änderungen (markiert mit *HEAD*)
- darunter die Änderung des Konflikte verursachenden Commits (markiert mit der *Commit-ID*)

- kompletter Abschnitt zwischen `<<<<` und `>>>>` muss jetzt von Hand gemerged werden
- löscht die Änderungen, die weg sollen und fasst alles so zusammen, dass es euch passt
- seht euch mit `git status` den Zwischenstand an

Was sehe ich da?

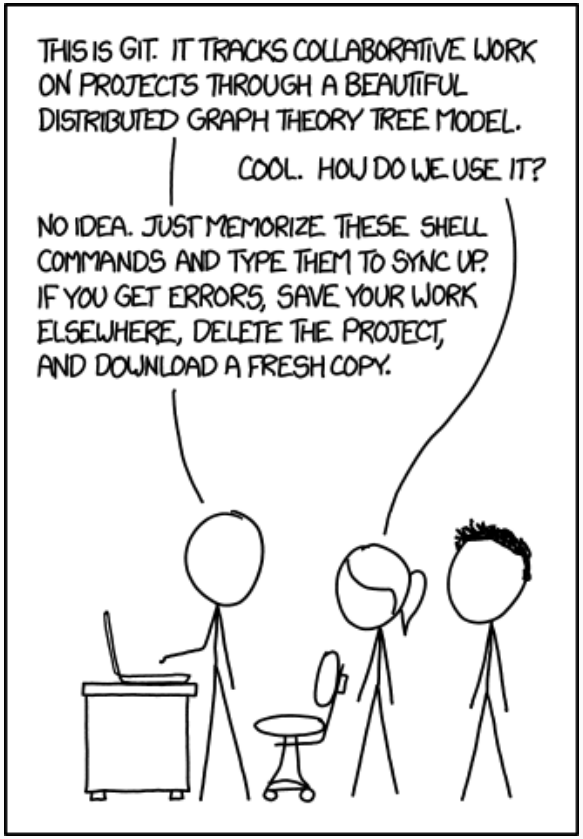
- oberhalb des ======= stehen eure Änderungen (markiert mit *HEAD*)
- darunter die Änderung des Konflikte verursachenden Commits (markiert mit der *Commit-ID*)

- kompletter Abschnitt zwischen <<<< und >>>> muss jetzt von Hand gemerged werden
- löscht die Änderungen, die weg sollen und fasst alles so zusammen, dass es euch passt
- seht euch mit `git status` den Zwischenstand an
- addet die „reparierte“ Datei
- commited das Ergebnis

Merge Konflikte sind kein Beinbruch!

Generell gilt: Ruhe bewahren!

- Merge Konflikte immer mit Sorgfalt beheben, besonders wenn große Code-Segmente betroffen sind!
- Ein Konflikt kann sich über mehrere Blöcke erstrecken
- schaut euch betroffene Dateien genau an!



Glückwunsch!

Ihr könnt jetzt:

- Euer Projekt mit `git` verwalten
- euer Repository mit GitHub über mehrere PCs und User synchronisieren
- Merge Konflikte beheben
- fremde Projekte klonen und mitmachen!

Ihr seid für eure ersten Schritte mit `git` gewappnet!

Doch mit git ist noch viel mehr möglich!

Werft zum Beispiel mal ein Blick auf folgende Dinge:

- die `.gitignore`-Datei
- Branches
- `git log`
  auch mit den Zusätzen
  `--oneline`
  `--graph`
- `git diff`
- `git pull --rebase`
- `git commit --amend`
- `git fetch` vs. `git pull`

Hilfe, so viele Befehle! Das merk ich mir nie!

Musst du auch nicht, es gibt ja Cheat Sheets:

- GitHub git Cheat Sheet Education
- Cheat Sheet von Tower
- und viele mehr - einfach mal googlen

Außerdem gibt es zahlreiche Guides und Einführungen im Netz:

- git - Der einfache Einstieg
- Git Explained: For Beginners
- und viele mehr, schaut einfach, was euch anspricht!

Tipp: Unbedingt über Good Practices informieren!

Ausblick

Nächster Termin:

kommt noch per Mail

Freut euch auf:

- Wie geht es weiter? - Ausblick auf kommende Semester
- Tipps zum Stundenplanbau
- Wie gehe ich mit dem Prüfungsamt um?
- Und...ein paar letzte Tipps bevor ihr in die Welt der höheren Semester entlassen werdet :)