Versionsverwaltung mit Git und GitLab

Andreas Poenicke, Daniel Hauck

21. Januar 2025

Inhaltsverzeichnis

1 Git

1.1 Initialisieren

```
~ $ mkdir example-project
~ $ cd example-project
~/example-project $ git init
Leeres Git-Repository in /home/daniel/example-project/.git/ initialisiert
```

1.2 Hilfe

Zu jedem Git-Befehl kann eine Hilfe abgerufen werden, z.B.

```
~/example-project $ git help init
GIT-INIT(1)                 Git Manual                 GIT-INIT(1)

NAME
       git-init - Create an empty Git repository or reinitialize an
       existing one

SYNOPSIS
       git init [-q | --quiet] [--bare] [--template=<template_directory>]
                 [--separate-git-dir <git dir>] [--object-format=<format>]
                 [-b <branch-name> | --initial-branch=<branch-name>]
                 [--shared[=<permissions>]] [directory]

DESCRIPTION
       This command creates an empty Git repository - basically a
       .git directory with subdirectories for objects, refs/heads,
       refs/tags, and template files. An initial branch without any
       commits will be Create (see the --initial-branch option
       below for its name).
...
```

Außerdem gibt es auch eine kurze Übersicht mit der Option `-h`

```
~/example-project $ git init -h
Verwendung: git init [-q | --quiet] [--bare] [--template=<Vorlagenverzeichnis>] [--shared
    ↪ [=<Berechtigungen>]] [<Verzeichnis>]

    --template <Vorlagenverzeichnis>
                          Verzeichnis, von welchem die Vorlagen verwendet werden
    --bare                ein Bare-Repository erstellen
    --shared[=<Berechtigungen>]
                          angeben, dass das Git-Repository mit mehreren Benutzern geteilt
    ↪ wird
    -q, --quiet           weniger Ausgaben
    --separate-git-dir <.git-Verzeichnis>
                          Git-Verzeichnis vom Arbeitsverzeichnis separieren
    -b, --initial-branch <Name>
                          den Namen des initialen Branches überschreiben
    --object-format <Hash>
                          den zu verwendenen Hash-Algorithmus angebe
```

1.3 Konfiguration

Übersicht über die bestehende Konfiguration

```
~/example-project $ git config --list
user.name=Daniel Hauck
user.email=daniel.hauck@kit.edu
```

```
...
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
~/example-project $ git config --list --show-scope
global  user.name=Daniel Hauck
global  user.email=daniel.hauck@kit.edu
...
local   core.repositoryformatversion=0
local   core.filemode=true
local   core.bare=false
local   core.logallrefupdates=true
~/example-project $ git config --list --show-origin
file:/home/daniel/.config/git/config      user.name=Daniel Hauck
file:/home/daniel/.config/git/config      user.email=daniel.hauck@kit.edu
...
file:.git/config        core.repositoryformatversion=0
file:.git/config        core.filemode=true
file:.git/config        core.bare=false
file:.git/config        core.logallrefupdates=true
```

Ohne `user.name` und `user.email` können keine Commits durchgeführt werden

1.4 Informationen über Repository

```
~/example-project $ git status
Auf Branch main

Noch keine Commits

nichts zu committen (erstellen/kopieren Sie Dateien und benutzen
Sie "git add" zum Versionieren)
```

1.5 Änderungen

Änderungen zur Staging-Area hinzufügen

```
~/example-project $ echo "Hallo" > foo.txt
~/example-project $ git status
Auf Branch main

Noch keine Commits

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
        foo.txt
```

```
nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
~/example-project $ git add foo.txt # oder für ganzes Verzeichnis: git add .
~/example-project $ git status
Auf Branch main

Noch keine Commits

Zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)
        neue Datei:     foo.txt
```

Änderungen comitten

```
~/example-project $ git commit
Hinweis: Warte auf das Schließen der Datei durch Ihren Editor...
...
```

Dies öffnet einen Texteditor, in dem dann eine Beschreibung eingegeben wird

```
1 Create foo.txt file
2 # Bitte geben Sie eine Commit-Beschreibung für Ihre Änderungen ein. Zeilen,
3 # die mit '#' beginnen, werden ignoriert, und eine leere Beschreibung
4 # bricht den Commit ab.
5 #
6 # Auf Branch main
7 #
8 # Initialer Commit
9 #
10 # Zum Commit vorgemerkte Änderungen:
11 #       neue Datei:     foo.txt
12 #
```

```
...
[main (Root-Commit) 2e18328] Create foo.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 foo.txt
```

Alternativ kann auch direkt die Nachricht mit der Option `-m` angegeben werden

```
~/example-project $ echo "Hello again" >> foo.txt
~/example-project $ git add foo.txt
~/example-project $ git commit -m "Say hello once more in foo.txt"
[main 7049182] Say hello once more in foo.txt
 1 file changed, 1 insertion(+)
```

Zudem können auch mit `git commit -a` automatisch alle Änderungen in die Stagin-Area übernommen werden. Dies ist aber gefährlich, da schnell ungewollte Änderungen übernommen werden, und sollte vermieden werden. Außerdem kann mit `git commit --amend` der letzte Commit bearbeitet werden.

Wenn bereits gepushte Commits geändert werden, führt das zu Konflikten.

1.5.1 Staging-Area

Allgemein kann der Prozess wie folgt zusammengefasst werden:

- `git add` überträgt Änderungen in Zwischenspeicher (Staging Area)
- Staging Area kann weiter verändert werden.
- `git commit` überträgt Änderungen aus Staging Area ins Repositorium
- Staging-Area wird dabei wieder geleert.
- Ermöglicht den „Commit“ in kleinen Einzelschritten aufzubauen

Mächtig um Änderungen an verschiedenen Stellen um Filesystem zusammenzufassen

1.6 Änderungen zurücknehmen

Änderungen an Dateien können mit `git restore` zurückgenommen werden [1].

```
~/example-project $ echo "BYE" > foo.txt
~/example-project $ git add foo.txt
~/example-project $ git status
~/example-project $ git status
Auf Branch main
Zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git restore --staged <Datei>..." zum Entfernen aus der Staging-Area)
        geändert:       foo.txt
~/example-project $ git restore --staged foo.txt
~/example-project $ git status
Auf Branch main
Änderungen, die nicht zum Commit vorgemerkt sind:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
  (benutzen Sie "git restore <Datei>...", um die Änderungen im Arbeitsverzeichnis zu
    ↪ verwerfen)
        geändert:       foo.txt

keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git commit -a")
~/example-project $ git restore foo.txt
~/example-project $ git status
Auf Branch main
nichts zu committen, Arbeitsverzeichnis unverändert
```

[1] `git restore` ist eine moderne Variante eines Teils der Funktionen von `git checkout`.

1.7 Dateien ignorieren

```
~/example-project $ echo "Test" > test.tmp
~/example-project $ git status
Auf Branch main
Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
        test.tmp

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
```

Datei `.gitignore` mit einem Texteditor bearbeiten

```
1 *.tmp
```

```
~/example-project $ git add .gitignore
~/example-project $ git commit -m 'Ignore temporary files'
[main 5c26ae3] Ignore temporary files
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
~/example-project $ git status
Auf Branch main
nichts zu committen, Arbeitsverzeichnis unverändert
~/example-project $ git status --ignored
Auf Branch main
Ignorierte Dateien:
  (benutzen Sie "git add -f <Datei>...", um die Änderungen zum Commit vorzumerken)
        test.tmp

nichts zu committen, Arbeitsverzeichnis unverändert
~/example-project $ git add test.tmp
Die folgenden Pfade werden durch eine Ihrer ".gitignore" Dateien ignoriert:
test.tmp
Hinweis: Nutzen Sie die Option -f, wenn sie wirklich hinzugefügt werden sollen.
Hinweis: Um diese Meldung abzuschalten, führen Sie folgenden Befehl aus:
Hinweis: "git config advice.addIgnoredFile false"
```

1.8 Dateien löschen

```
~/example-project $ rm foo.txt
~/example-project $ git add foo.txt
~/example-project $ git commit -m 'Remove foo.txt'
[main f39f021] Remove foo.txt
 1 file changed, 2 deletions(-)
 delete mode 100644 foo.txt
```

1.9 Protokoll anzeigen

```
~/example-project $ git log
commit f39f021a46debd9cbda007e54bdce2cdc60b7ccf (HEAD -> main)
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Mon Nov 25 15:58:12 2024 +0100

    Remove foo.txt

commit 5c26ae3f1ccd770412e3f90047b3ea99b448a9c2
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Mon Nov 25 15:55:55 2024 +0100

    Ignore temporary files

commit 704918228d3baa5938be9d4989ed638579072aa4
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Thu Nov 21 16:46:20 2024 +0100

    Say hello once more in foo.txt

commit 2e18328364dd65484bb17875ec567297dfb114db
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Thu Nov 21 16:10:44 2024 +0100

    Create foo.txt file
~/example-project $ git log -2
commit f39f021a46debd9cbda007e54bdce2cdc60b7ccf (HEAD -> main)
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Mon Nov 25 15:58:12 2024 +0100

    Remove foo.txt

commit 5c26ae3f1ccd770412e3f90047b3ea99b448a9c2
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Mon Nov 25 15:55:55 2024 +0100

    Ignore temporary files
~/example-project $ git log --oneline
f39f021 (HEAD -> main) Remove foo.txt
5c26ae3 Ignore temporary files
7049182 Say hello once more in foo.txt
2e18328 Create foo.txt file
```

Für komplexere Situationen (z.B. mit Branches) können auch `git log --graph` und das graphische Werkzeug `gitk` nützlich sein.

```
~/example-project $ git show 7049182
commit 704918228d3baa5938be9d4989ed638579072aa4
Author: Daniel Hauck <daniel.hauck@kit.edu>
Date:   Thu Nov 21 16:46:20 2024 +0100
```

```
    Say hello once more in foo.txt

diff --git a/foo.txt b/foo.txt
index b5fc21b..1f3dc8f 100644
--- a/foo.txt
+++ b/foo.txt
@@ -1 +1,2 @@
 Hallo
+Hello again
~/example-project $ git show 7049182 -- foo.txt
...
```

1.10 Unterschiede anzeigen

```
~/example-project $ git diff 5c26ae3..HEAD # alternativ git diff HEAD~1..HEAD
diff --git a/foo.txt b/foo.txt
deleted file mode 100644
index 1f3dc8f..0000000
--- a/foo.txt
+++ /dev/null
@@ -1,2 +0,0 @@
-Hallo
-Hello again
~/example-project $ touch bar.txt
~/example-project $ echo "*.bak" >>.gitignore
~/example-project $ git add .gitignore
~/example-project $ echo "*.png" >>.gitignore
~/example-project $ git diff
diff --git a/.gitignore b/.gitignore
index 0be9a7f..36bd8f4 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,2 +1,3 @@
 *.tmp
 *.bak
+*.png
~/example-project $ git diff --cached $ Änderungen in der Staging-Area
diff --git a/.gitignore b/.gitignore
index 1944fd6..0be9a7f 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1 +1,2 @@
 *.tmp
+*.bak
```

1.11 Zeitreise

Einzelne Dateien

```
~/example-project $ git restore -s 5c26ae3 foo.txt
~/example-project $ cat foo.txt
Hallo
Hello again
~/example-project $ git commit -m 'Restore old foo.txt'
```

Ganzes Verzeichnis

> Wenn ein Commit ausgecheckt wird, ist der `HEAD` *losgelöst*. Dies muss explizit mit `-d` ausgewählt werden.[3]

```
~/example-project $ git switch -d 5c26ae3
~/example-project $ git log --all --oneline
236b815 (main) Restore old foo.txt
f39f021 Remove foo.txt
5c26ae3 (HEAD) Ignore temporary files
7049182 Say hello once more in foo.txt
2e18328 Create foo.txt file
~/example-project $ git status
HEAD losgelöst bei 5c26ae3
nichts zu committen, Arbeitsverzeichnis unverändert
~/example-project $ cat foo.txt
Hallo
Hello again
~/example-project $ git switch main # zurück zum main Branch
```

1.12 Kennzeichner (Tags)

```
~/example-project $ git tag second
~/example-project $ git tag first HEAD~1
~/example-project $ git log --oneline
236b815 (HEAD -> main, tag: second) Restore old foo.txt

f39f021 (tag: first) Remove foo.txt
5c26ae3 Ignore temporary files
7049182 Say hello once more in foo.txt
2e18328 Create foo.txt file
~/example-project $ git tag
first
second
```

Tags sind wie Commits Referenzen und können daher überall auch als solche verwenden

[3] Eine ältere Variante von `git switch` ist `git checkout`. Dort wird dies nicht explizit angegeben, weshalb man dabei noch vorsichtiger sein muss.

```
~/example-project $ git diff first..second
diff --git a/foo.txt b/foo.txt
new file mode 100644
index 0000000..1f3dc8f
--- /dev/null
+++ b/foo.txt
@@ -0,0 +1,2 @@
+Hallo
+Hello again
~/example-project $ git switch -d first
Hinweis: Wechsle zu 'first'.

Sie befinden sich im Zustand eines 'losgelösten HEAD'. Sie können sich
umschauen, experimentelle Änderungen vornehmen und diese committen, und
Sie können alle möglichen Commits, die Sie in diesem Zustand machen,
ohne Auswirkungen auf irgendeinen Branch verwerfen, indem Sie zu einem
anderen Branch wechseln.

Wenn Sie einen neuen Branch erstellen möchten, um Ihre erstellten Commits
zu behalten, können Sie das (jetzt oder später) durch Nutzung von
'switch' mit der Option -c tun. Beispiel:

  git switch -c <neuer-Branchname>

Oder um diese Operation rückgängig zu machen:
  git switch -

Sie können diesen Hinweis ausschalten, indem Sie die Konfigurationsvariable
'advice.detachedHead' auf 'false' setzen.

HEAD ist jetzt bei f39f021 Remove foo.txt
~/example-project $ git switch main
Vorherige Position von HEAD war f39f021 Remove foo.txt
Zu Branch 'main' gewechselt
```

Tags können auch gelöscht werden

```
~/example-project $ git tag -d second
Tag 'second' gelöscht (war 236b815)
~/example-project $ git log --oneline
236b815 (HEAD -> main) Restore old foo.txt
f39f021 (tag: first) Remove foo.txt
5c26ae3 Ignore temporary files
7049182 Say hello once more in foo.txt
2e18328 Create foo.txt file
```

1.13 Ganze Commits zurücknehmen

Mit neuem Commit (bevorzugt, da Geschichte erhalten bleibt)

```
~/example-project $ git revert 7049182 # git revert HEAD für letzten Commit
```

```
[main 939e6c8] Revert "Say hello once more in foo.txt"
 1 file changed, 1 deletion(-)
~/example-project $ git log --oneline
939e6c8 (HEAD -> main) Revert "Say hello once more in foo.txt"
236b815 Restore old foo.txt
f39f021 (tag: first) Remove foo.txt
5c26ae3 Ignore temporary files
7049182 Say hello once more in foo.txt
2e18328 Create foo.txt file
```

Wenn seit dem Commit inkompatible Änderungen aufgetreten sind, kann es hier zu Konflikten, die ein Mergen notwendig machen.

Das Arbeitsverzeichnis darf hierbei keine Änderungen enthalten. Dies kann z.B. durch `git stash` erzielt werden, wenn die aktuellen Änderungen nicht verloren gehen sollen.

Änderungen verwerfen

```
~/example-project $ git reset HEAD~ # äquivalent zu git reset --mixed
Nicht zum Commit vorgemerkte Änderungen nach Zurücksetzung:
M       foo.txt
~/example-project $ git status
Auf Branch main
Änderungen, die nicht zum Commit vorgemerkt sind:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
  (benutzen Sie "git restore <Datei>...", um die Änderungen im Arbeitsverzeichnis zu
    ↪ verwerfen)
        geändert:       foo.txt

keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git commit -a")
~/example-project $ git log --oneline
236b815 (HEAD -> main) Restore old foo.txt
f39f021 (tag: first) Remove foo.txt
5c26ae3 Ignore temporary files
7049182 Say hello once more in foo.txt
2e18328 Create foo.txt file
```

Alternativ existieren auch `git reset --soft` (behält die Änderungen in der Staging-Area) und `git reset --hard` (verwirft auch die Änderungen im aktuellen Verzeichnis)

Nie Commits verwerfen, die schon an ein Remote-Repository übertragen wurden! Besser: `git revert`

Direkt nach dem Reset kann mit `git reset ORIG_HEAD` der Reset auch wieder rückgängig gemacht werden.

2 GitLab

2.1 Authentifizierung

2.1.1 Token

```
~/example-project $ cd
~ $ git config --global credential.helper=manager # Sollte Standard sein
~ $ git config --global credential.https://gitlab.kit.edu.username YOURUSERNAME
~ $ git config --global credential.https://gitlab.kit.edu.helper=manager-core
```

In Gitlab

- Account → Edit profile
- Acess Tokens → Personal Access Tokens → Add new token
- personal access token → Copy (Achtung später nicht mehr abrufbar)

2.1.2 SSH-Key

```
~ $ ssh-keygen -t ed25519
```

Immer ein Passwort vergeben!

In Gitlab

- Account → Edit profile
- SSH Keys → Add new key
- Inhalt von `~/.ssh/id_ed25519.pub` als Key einfügen

Hier den öffentlichen Key verwenden, also die auf `.pub` endende Datei

```
~ $ ssh -T git@gitlab.kit.edu # Password von Key
Enter passphrase for key '/home/daniel/.ssh/id_ed25519':
Welcome to GitLab, @daniel.hauck!
~ $ ssh-add # Password von Key
Enter passphrase for key '/home/daniel/.ssh/id_ed25519':
Identity added: /home/daniel/.ssh/id_ed25519 (daniel@hostname)
~ $ ssh -T git@gitlab.kit.edu # kein Password
Welcome to GitLab, @daniel.hauck!
```

2.2 Repository erstellen

- "+"→ New project/repository
- Create blank project
- Project name: example-project
- Visibility Level: Private
- Create project → öffnet Projekt
- Rechte Seite → Code → Clone with SSH (SSH-Key) / Clone with HTTPS (Token) → *kopieren*

```
~ $ git clone git@gitlab.kit.edu:daniel.hauck/git-example.git # hier die kopierte URL
    ↪ einsetzen
Klone nach 'git-example' ...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Empfange Objekte: 100% (3/3), fertig.
~ $ cd git-example
~/git-example $ git remote -v
origin  git@gitlab.kit.edu:daniel.hauck/git-example.git (fetch)
origin  git@gitlab.kit.edu:daniel.hauck/git-example.git (push)
```

2.3 Änderungen synchronisieren

```
~/git-example $ git pull # holt Änderungen vom Remote
Bereits aktuell.
~/git-example $ echo "Hallo" > foo.txt
~/git-example $ git add foo.txt
~/git-example $ git commit -m "Say hallo to foo"
[main 83461a4] Say hallo to foo
 1 file changed, 1 insertion(+)
 create mode 100644 foo.txt
~/git-example $ git push
Objekte aufzählen: 4, fertig.
Zähle Objekte: 100% (4/4), fertig.
Delta-Kompression verwendet bis zu 16 Threads.
Komprimiere Objekte: 100% (2/2), fertig.
Schreibe Objekte: 100% (3/3), 284 Bytes | 284.00 KiB/s, fertig.
Gesamt 3 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
To gitlab.kit.edu:daniel.hauck/git-example.git
   4de79e6..83461a4  main -> main
```

Wenn auch Tags synchronisiert werden sollen, kann das mit `git pull --tags` und `git push --tags` erreicht werden.

2.4 Konflikte

```
~/git-example $ cd ..
~ $ git clone git@gitlab.kit.edu:daniel.hauck/git-example.git git-example2
~ $ cd git-example2
~/git-example2 $ echo "Foo" > hello.txt
~/git-example2 $ git add hello.txt
~/git-example2 $ git commit -m "Use Foo for hello"
[main b48de63] Use Foo for hello
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
~/git-example2 $ git push
Objekte aufzählen: 4, fertig.
Zähle Objekte: 100% (4/4), fertig.
Delta-Kompression verwendet bis zu 16 Threads.
Komprimiere Objekte: 100% (2/2), fertig.
Schreibe Objekte: 100% (3/3), 313 Bytes | 156.00 KiB/s, fertig.
Gesamt 3 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
To gitlab.kit.edu:daniel.hauck/git-example.git
   83461a4..b48de63  main -> main
~/git-example2 $ cd ../git-example
~/git-example $ echo "Bar" > hello.txt
~/git-example $ git add hello.txt
~/git-example $ git commit -m "Use Bar for hello"
[main 8b5a5c7] Use Bar for hello
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
~/git-example $ git pull
Hinweis: Es wird davon abgeraten zu Pullen, ohne anzugeben, wie mit abweichenden
Hinweis: Branches umgegangen werden soll. Sie können diese Nachricht unterdrücken,
Hinweis: indem Sie einen der folgenden Befehle ausführen, bevor der nächste Pull
Hinweis: ausgeführt wird:
Hinweis:
Hinweis:   git config pull.rebase false  # Merge (Standard-Strategie)
Hinweis:   git config pull.rebase true   # Rebase
Hinweis:   git config pull.ff only       # ausschließlich Vorspulen
Hinweis:
Hinweis: Sie können statt "git config" auch "git config --global" nutzen, um
Hinweis: einen Standard für alle Repositories festzulegen. Sie können auch die
Hinweis: Option --rebase, --no-rebase oder --ff-only auf der Kommandozeile nutzen,
Hinweis: um das konfigurierte Standardverhalten pro Aufruf zu überschreiben.
KONFLIKT (hinzufügen/hinzufügen): Merge-Konflikt in hello.txt
automatischer Merge von hello.txt
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das
    ↪ Ergebnis.
~/git-example $ git config pull.rebase false # optional, aber empfohlen
~/git-example $ git status
Auf Branch main
Ihr Branch und 'origin/main' sind divergiert,
und haben jeweils 1 und 1 unterschiedliche Commits.
  (benutzen Sie "git pull", um Ihren Branch mit dem Remote-Branch zusammenzuführen)
```

```
Sie haben nicht zusammengeführte Pfade.
  (beheben Sie die Konflikte und führen Sie "git commit" aus)
  (benutzen Sie "git merge --abort", um den Merge abzubrechen)

Nicht zusammengeführte Pfade:
  (benutzen Sie "git add/rm <Datei>...", um die Auflösung zu markieren)
        von beiden hinzugefügt: hello.txt

keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git commit -a")
~/git-example $ git log --oneline --graph --all
* 8b5a5c7 (HEAD -> main) Use Bar for hello
| * b48de63 (origin/main, origin/HEAD) Use Foo for hello
|/
* 83461a4 Say hallo to foo
* 4de79e6 Initial commit
```

Die Konflikte müssen jetzt zunächst behoben werden, indem die Datei in einem Texteditor bearbeitet wird.

```
1 <<<<<<< HEAD
2 Bar
3 =======
4 Foo
5 >>>>>>> b48de633399e50883ff65724ccebb86a1ea52b0c
```

Hier nehmen wir an, dass der Inhalt wie folgt sein soll

```
1 FooBar
```

Jetzt kann ein Commit erstellt werden, um den Merge abzuschließen

```
~/git-example $ git add hello.txt
~/git-example $ git commit -m "Use FooBar for hello"
~/git-example $ git push
Objekte aufzählen: 9, fertig.
Zähle Objekte: 100% (9/9), fertig.
Delta-Kompression verwendet bis zu 16 Threads.
Komprimiere Objekte: 100% (4/4), fertig.
Schreibe Objekte: 100% (6/6), 521 Bytes | 521.00 KiB/s, fertig.
Gesamt 6 (Delta 2), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
To gitlab.kit.edu:daniel.hauck/git-example.git
   b48de63..806f313  main -> main
~/git-example $ git log --oneline --graph --all
*   806f313 (HEAD -> main, origin/main, origin/HEAD) Use FooBar for hello
|\
| * b48de63 Use Foo for hello
* | 8b5a5c7 Use Bar for hello
|/
* 83461a4 Say hallo to foo
* 4de79e6 Initial commit
```

2.5 Lokal erstelltes Reposiory hochladen

```
~/git-example $ cd ../example-project
~/example-project $ git remote add origin git@gitlab.kit.edu:daniel.hauck/example-project.
    ↪ git
~/example-project $ git push
fatal: Der aktuelle Branch main hat keinen Upstream-Branch.
Um den aktuellen Branch zu versenden und den Remote-Branch
als Upstream-Branch zu setzen, benutzen Sie

    git push --set-upstream origin main
~/example-project $ git push --set-upstream origin main
Objekte aufzählen: 14, fertig.
Zähle Objekte: 100% (14/14), fertig.
Delta-Kompression verwendet bis zu 16 Threads.
Komprimiere Objekte: 100% (8/8), fertig.
Schreibe Objekte: 100% (14/14), 1.29 KiB | 439.00 KiB/s, fertig.
Gesamt 14 (Delta 1), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
remote:
remote:
remote: The private project daniel.hauck/example-project was successfully created.
remote:
remote: To configure the remote, run:
remote:   git remote add origin git@gitlab.kit.edu:daniel.hauck/example-project.git
remote:
remote: To view the project, visit:
remote:   https://gitlab.kit.edu/daniel.hauck/example-project
remote:
remote:
remote:
To gitlab.kit.edu:daniel.hauck/example-project.git
 * [new branch]      main -> main
Branch 'main' folgt nun Remote-Branch 'main' von 'origin'.
```

Gitlab erstellt hierbei automatisch ein privates Repository.

Wenn das Repository zuvor über das Webinterface erstellt wird muss es unbedingt leer sein. Dazu muss die Erstellung eine Readme-Datei deaktiviert werden.