Fachschaft Elektro- und Informationstechnik

Revisionskontrolle mit GIT

Benjamin Berg

4. Dezember 2012

Inhalt

Vorteile von Versionierung

- Speicherung der Geschichte
- Dokumentation der Änderungen
- Kollaboration
- Sicherheitskopien

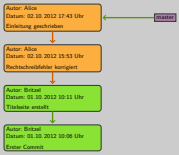
Warum möchten wir überhaupt Versionieren? Zu jedem der Punkte (teilweise nur freie Software):

- Autor ist bekannt → Rückfragen wenn etwas nicht funktioniert
- Autor ist bekannt → Lizenzprobleme können geklärt werden
- Geschichte: Fehlersuche durch vergleich mit alten Versionen (seit wann existiert der Fehler? wodurch ist er entstanden?)
- Autoren können Weltweit verteilt Arbeiten
- Sicherheitskopien: Man verzettelt sich nicht mit vielen unterschiedlichen Versionen

Wichtige Begriffe

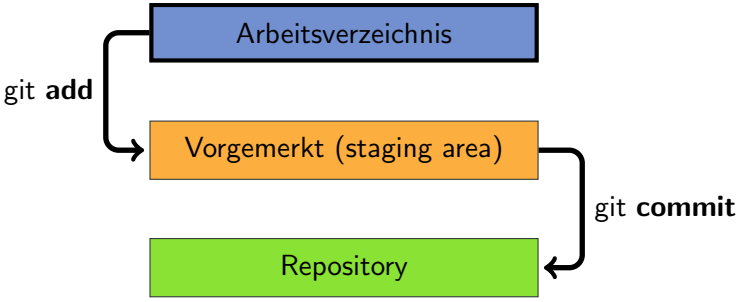
repository	Das Archiv
commit	Ein bestimmter Zustand
branch	Verzweigung, Entwicklungslinie
head	Referenz auf den neuesten „commit“ eines „branch“
master	Standard „branch“
staging area	Speicherort für vorgemerkte Änderungen

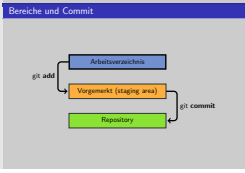
commit	Englisch: festlegen
branch	Vom englischen Wort für „Ast“; Nach einer Verzweigung gibt es mehrere Äste
staging area	Manchmal auch als „index“ Bezeichnet



- Referenz auf den/die Vorgänger gehört zum „commit“
- „master“ Zeigt auf den zuletzt erstellten „commit“
- Erster Commit hat keinen Vorgänger

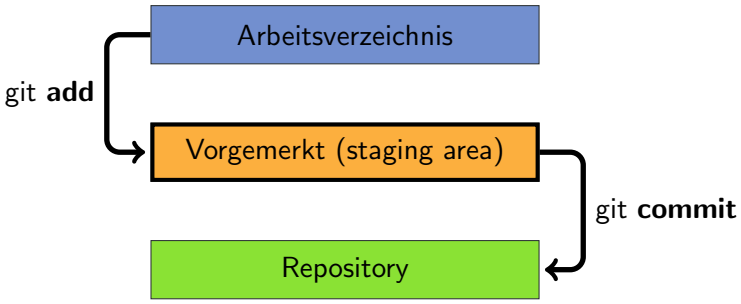
Bereiche und Commit

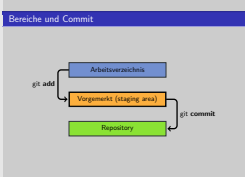




Hier sollte auf die einzelnen Bereiche eingegangen und deren Eigenschaften eingegangen werden.

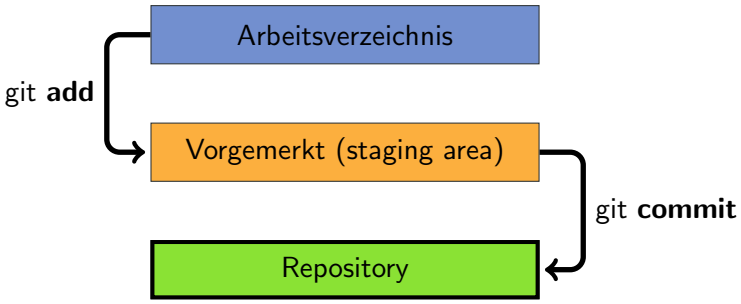
Bereiche und Commit

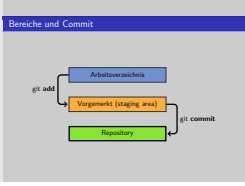




Hier sollte auf die einzelnen Bereiche eingegangen und deren Eigenschaften eingegangen werden.

Bereiche und Commit





Hier sollte auf die einzelnen Bereiche eingegangen und deren Eigenschaften eingegangen werden.

git **add**

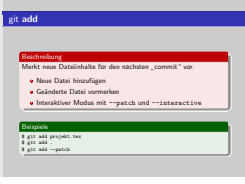
Beschreibung

Merkt neue Dateiinhalte für den nächsten „commit“ vor.

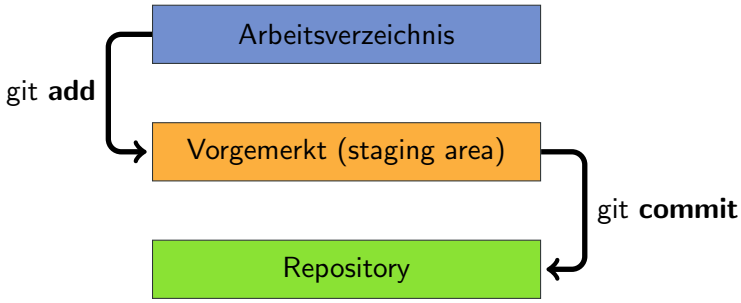
- Neue Datei hinzufügen
- Geänderte Datei vormerken
- Interaktiver Modus mit `--patch` und `--interactive`

Beispiele

```
$ git add projekt.tex
$ git add .
$ git add --patch
```

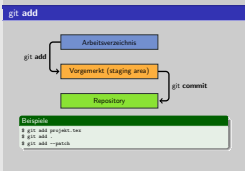



git add



Beispiele

```
$ git add projekt.tex
$ git add .
$ git add --patch
```

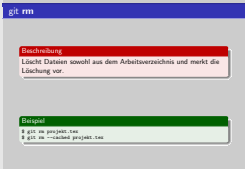
git rm

Beschreibung

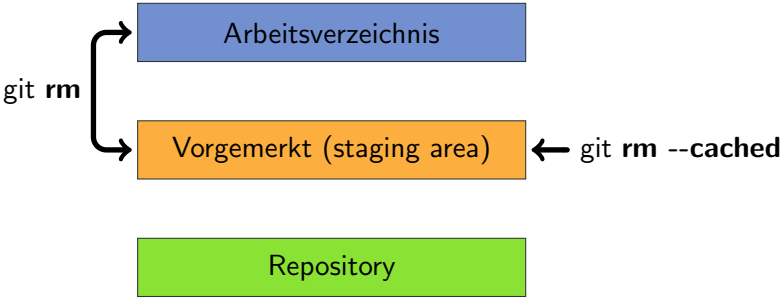
Löscht Dateien sowohl aus dem Arbeitsverzeichnis und merkt die Löschung vor.

Beispiel

```
$ git rm projekt.tex
$ git rm --cached projekt.tex
```

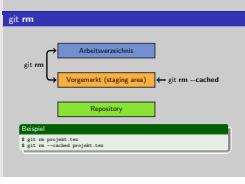



git rm



Beispiel

```
$ git rm projekt.tex
$ git rm --cached projekt.tex
```

git **reset**

Beschreibung

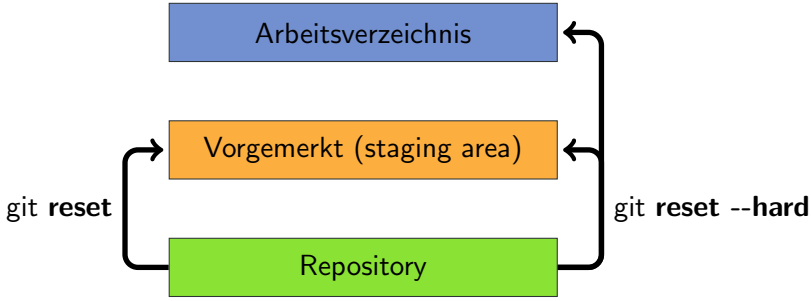
Setzt Dateien in ihren Ursprungszustand zurück.

- Verwerfen von vorgemerkten Änderungen
- Verwerfen von Änderungen im Arbeitsverzeichnis

Beispiel

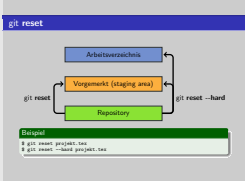
```
$ git reset projekt.tex
$ git reset --hard projekt.tex
```


git reset



Beispiel

```
$ git reset projekt.tex
$ git reset --hard projekt.tex
```

git **diff**

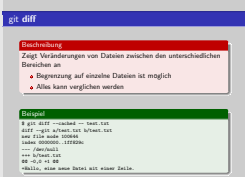
Beschreibung

Zeigt Veränderungen von Dateien zwischen den unterschiedlichen Bereichen an

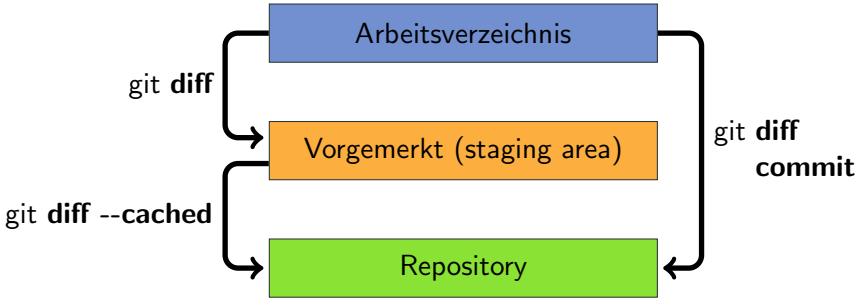
- Begrenzung auf einzelne Dateien ist möglich
- Alles kann verglichen werden

Beispiel

```
$ git diff --cached -- test.txt
diff --git a/test.txt b/test.txt
new file mode 100644
index 0000000..1ff829c
--- /dev/null
+++ b/test.txt
@@ -0,0 +1 @@
+Hallo, eine neue Datei mit einer Zeile.
```

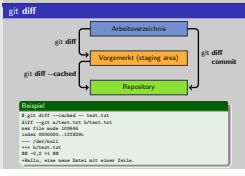



git diff



Beispiel

```
$ git diff --cached -- test.txt
diff --git a/test.txt b/test.txt
new file mode 100644
index 0000000..1ff829c
--- /dev/null
+++ b/test.txt
@@ -0,0 +1 @@
+Hallo, eine neue Datei mit einer Zeile.
```

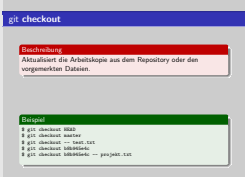
git checkout

Beschreibung

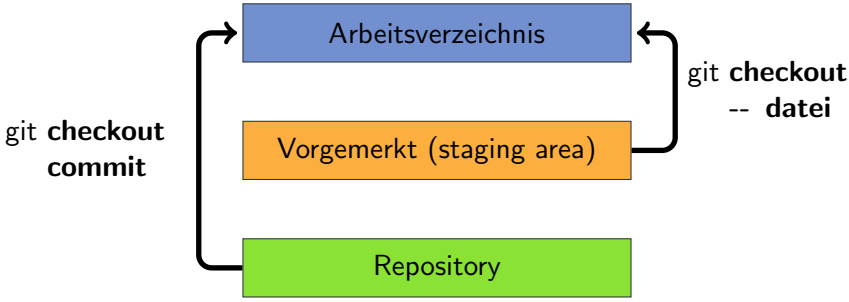
Aktualisiert die Arbeitskopie aus dem Repository oder den vorgemerkten Dateien.

Beispiel

```
$ git checkout HEAD
$ git checkout master
$ git checkout -- test.txt
$ git checkout b8b945e4c
$ git checkout b8b945e4c -- projekt.txt
```

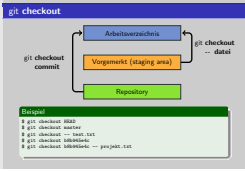



git checkout



Beispiel

```
$ git checkout HEAD
$ git checkout master
$ git checkout -- test.txt
$ git checkout b8b945e4c
$ git checkout b8b945e4c -- projekt.txt
```

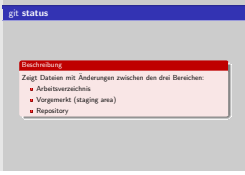



git **status**

Beschreibung

Zeigt Dateien mit Änderungen zwischen den drei Bereichen:

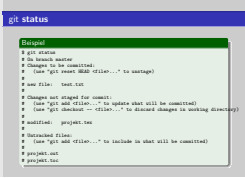
- Arbeitsverzeichnis
- Vorgemerkt (staging area)
- Repository



git **status**

Beispiel

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   test.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   projekt.tex
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# projekt.out
# projekt.toc
```

.gitignore File

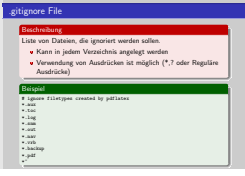
Beschreibung

Liste von Dateien, die ignoriert werden sollen.

- Kann in jedem Verzeichnis angelegt werden
- Verwendung von Ausdrücken ist möglich (*,? oder Reguläre Ausdrücke)

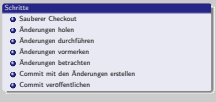
Beispiel

```
# ignore filetypes created by pdflatex
*.aux
*.toc
*.log
*.snm
*.out
*.nav
*.vrb
*.backup
*.pdf
*~
```

Schritte

1. Sauberer Checkout
2. Änderungen holen
3. Änderungen durchführen
4. Änderungen vormerken
5. Änderungen betrachten
6. Commit mit den Änderungen erstellen
7. Commit veröffentlichen



- Keine Doppelte Arbeit machen
- Nichts aus versehen committen.
- Einfach mit einem Texteditor
- git add
- Eigentlichen commit erstellen, der nicht mehr Veränderbar ist.
  Danach ist der Checkout wieder Sauber (wenn alles committed wurde)

Schritte

1. Sauberer Checkout	**git status**
2. Änderungen holen	**git pull**
3. Änderungen durchführen	
4. Änderungen vormerken	**git add**
5. Änderungen betrachten	**git diff --cached**
6. Commit mit den Änderungen erstellen	**git commit**
7. Commit veröffentlichen	**git push**

- Keine Doppelte Arbeit machen
- Nichts aus versehen committen.
- Einfach mit einem Texteditor
- git add
- Eigentlichen commit erstellen, der nicht mehr Veränderbar ist.
  Danach ist der Checkout wieder Sauber (wenn alles committed wurde)

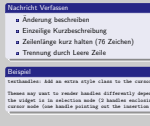
Nachricht Verfassen

- Änderung beschreiben
- Einzeilige Kurzbeschreibung
- Zeilenlänge kurz halten (76 Zeichen)
- Trennung durch Leere Zeile

Beispiel

```
texthandles: Add an extra style class to the cursor-mode handle

Themes may want to render handles differently depending on whether
the widget is in selection mode (2 handles enclosing a selection) or
cursor mode (one handle pointing out the insertion cursor).
```

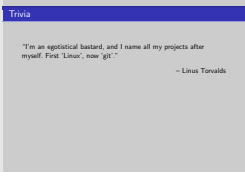



- Änderungen müssen beschrieben werden
- Präsens benutzen und Änderung beschreiben **nicht Handlungen**
- Längenregeln wegen 80 Zeichen Terminals üblich
- Wichtigste Regel: Leere Zeile
- Zum Beispiel:
  - Keyword um zu Zeigen wo die Änderung war.
  - Kurze Beschreibung der Änderung
  - Mehrzeilige Begründung für die Änderung.

Trivia

"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'."

– Linus Torvalds



- „git" beschreibt eher die Software selber. Das Datenbankformat ist extrem einfach/dumm gehalten.
- Völlig irrelevant: Es gibt ein Monty Python Script mit Mr. und Mrs. Git
- Andere Backronyms: "Global Information Tracker"; "Goddamn Idiotic Truckload of shit"

Trivia

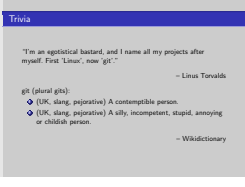
"I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'."

– Linus Torvalds

git (plural gits):

1. (UK, slang, pejorative) A contemptible person.
2. (UK, slang, pejorative) A silly, incompetent, stupid, annoying or childish person.

– Wikidictionary



- „git" beschreibt eher die Software selber. Das Datenbankformat ist extrem einfach/dumm gehalten.
- Völlig irrelevant: Es gibt ein Monty Python Script mit Mr. und Mrs. Git
- Andere Backronyms: "Global Information Tracker"; "Goddamn Idiotic Truckload of shit"

Kollaboration mit GIT

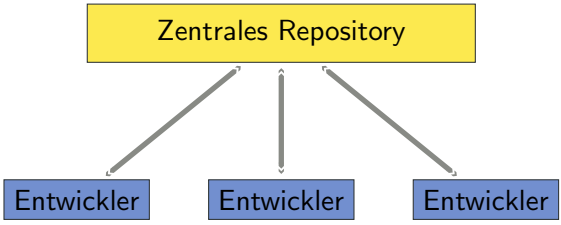
- Unterschiedliche Modelle sind möglich
- GIT nur ein Speicher und Transport

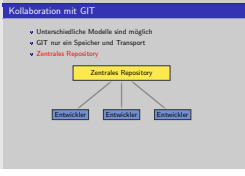
Kollaboration mit GIT
- Unterschiedliche Modelle sind möglich
- GIT nur ein Speicher und Transport

Im Linux Kernel werden Patches oft per E-Mail verschickt

Kollaboration mit GIT

- Unterschiedliche Modelle sind möglich
- GIT nur ein Speicher und Transport
- Zentrales Repository

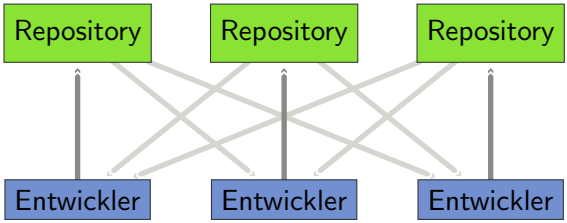


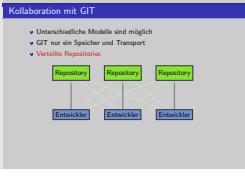


Im Linux Kernel werden Patches oft per E-Mail verschickt

Kollaboration mit GIT

- Unterschiedliche Modelle sind möglich
- GIT nur ein Speicher und Transport
- Verteilte Repositories

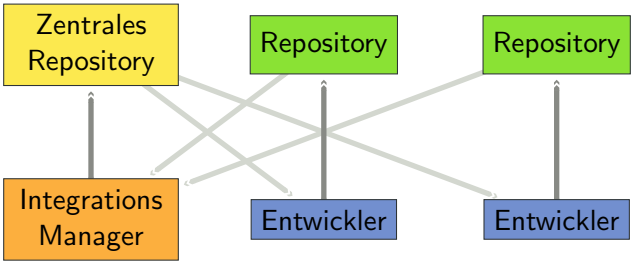


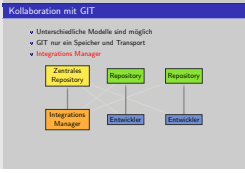


Im Linux Kernel werden Patches oft per E-Mail verschickt

Kollaboration mit GIT

- Unterschiedliche Modelle sind möglich
- GIT nur ein Speicher und Transport
- Integrations Manager

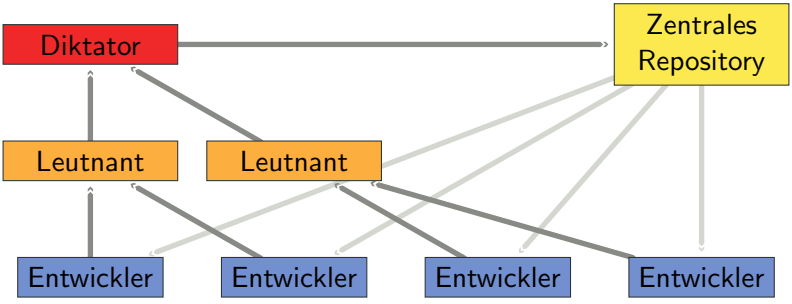


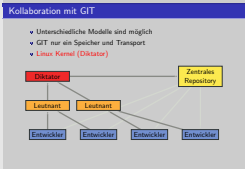


Im Linux Kernel werden Patches oft per E-Mail verschickt

Kollaboration mit GIT

- Unterschiedliche Modelle sind möglich
- GIT nur ein Speicher und Transport
- Linux Kernel (Diktator)

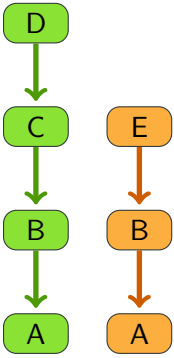




Im Linux Kernel werden Patches oft per E-Mail verschickt

Ursprungszustand

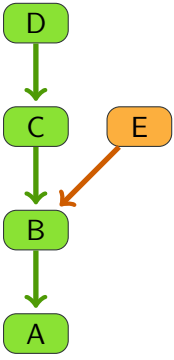
- Konflikte Vermeiden
- Zwei unterschiedliche Repositories
- Commits C,D und E sind unabhängige Änderungen
- Notwendigkeit der Zusammenführung

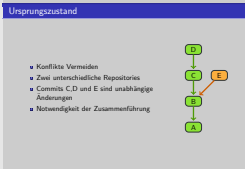


Wir haben zwei repositories mit anderen „master“ branchen. Diese müssen zusammengeführt werden.
Da Commits B jeweils gemeinsam sind, ist es möglich die als voneinander abhängig darzustellen (zweite Folie)

Ursprungszustand

- Konflikte Vermeiden
- Zwei unterschiedliche Repositories
- Commits C,D und E sind unabhängige Änderungen
- Notwendigkeit der Zusammenführung





Wir haben zwei repositories mit anderen „master“ branchen. Diese müssen zusammengeführt werden.
Da Commits B jeweils gemeinsam sind, ist es möglich die als voneinander abhängig darzustellen (zweite Folie)

git **init**

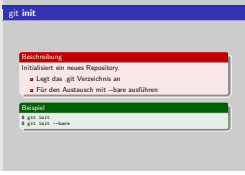
Beschreibung

Initialisiert ein neues Repository.

- Legt das .git Verzeichnis an
- Für den Austausch mit --bare ausführen

Beispiel

```
$ git init
$ git init --bare
```

Ein --bare Repository könnte zum Beispiel auf einem Server oder einem USB-Stick für den Austausch liegen.

git **push**

Beschreibung

Veränderungen in ein anderes Repository kopieren

- Fügt Commits in das andere Repository ein.
- Aktuallisiert die „branches“ (master).
- Funktioniert nur bei „fast forward“.

Beispiel

```
$ git push
$ git push /path/to/repo
$ git push ssh://alice@fachschaft.etec.uni-karlsruhe.de/home/git/repo.git
```


Ein --bare Repository könnte zum Beispiel auf einem Server oder einem USB-Stick für den Austausch liegen.

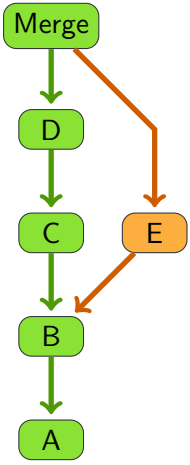
Fast forward: Seit dem letzten „pull“ hat sich nichts mehr auf dem Server geändert.

git **pull**

Beschreibung

Holt Änderungen von einem anderen Repository und versucht sie einzuspielen.

- Zuerst Änderungen holen (git **pull**)
- Dann Änderungen einspielen (git **merge**)
- Commit zum Zusammenführen kann notwendig werden



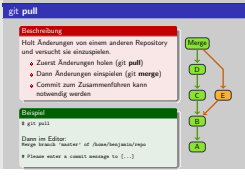
Beispiel

```
$ git pull
```

Dann im Editor:

```
Merge branch 'master' of /home/benjamin/repo

# Please enter a commit message to [...]
```

git pull

Konfliktfall

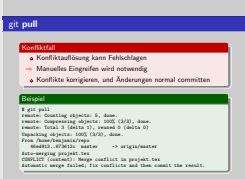
- Konfliktauflösung kann Fehlschlagen

⇒ Manuelles Eingreifen wird notwendig

- Konflikte korrigieren, und Änderungen normal committen

Beispiel

```
$ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /home/benjamin/repo
   46ed813..673612c  master     -> origin/master
Auto-merging projekt.tex
CONFLICT (content): Merge conflict in projekt.tex
Automatic merge failed; fix conflicts and then commit the result.
```

Es ist ein Fehler aufgetreten, den git nicht selber auflösen konnte

git **pull**

Konfliktfall

- Konfliktauflösung kann Fehlschlagen

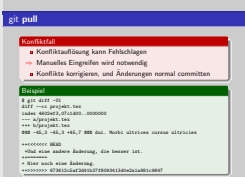
⇒ Manuelles Eingreifen wird notwendig

- Konflikte korrigieren, und Änderungen normal committen

Beispiel

```
$ git diff -U1
diff --cc projekt.tex
index 4602ef3,07c1d00..0000000
--- a/projekt.tex
+++ b/projekt.tex
@@@ -45,3 -45,3 +45,7 @@@ dui. Morbi ultrices cursus ultricies

++<<<<<<< HEAD
 +Und eine andere Änderung, die besser ist.
++=======
+ Hier noch eine Änderung.
++>>>>>>> 673612c5af2d41b27f8093413d0e2a1a881c9847
```

Das diff zeigt die von git eingefügten Zeilen. Beide Versionen die im Konflikt zueinander stehen sind zu sehen. (-U1 heißt nur eine Zeile aus der Umgebung, damit das Beispiel hier hin passt.)

Git zeigt hier die Differenz zu beiden alten Versionen. Oben ist unsere eigene, unten die des Anderen.

git **pull**

Konfliktfall

- Konfliktauflösung kann Fehlschlagen

⇒ Manuelles Eingreifen wird notwendig

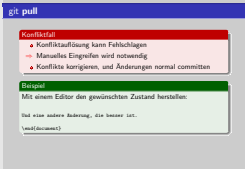
- Konflikte korrigieren, und Änderungen normal committen

Beispiel

Mit einem Editor den gewünschten Zustand herstellen:

```
Und eine andere Änderung, die besser ist.

\end{document}
```

Wir Editieren es einfach

git pull

Konfliktfall

- Konfliktauflösung kann Fehlschlagen

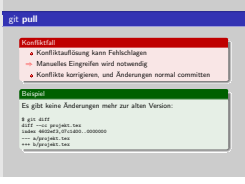
⇒ Manuelles Eingreifen wird notwendig

- Konflikte korrigieren, und Änderungen normal committen

Beispiel

Es gibt keine Änderungen mehr zur alten Version:

```
$ git diff
diff --cc projekt.tex
index 4602ef3,07c1d00..0000000
--- a/projekt.tex
+++ b/projekt.tex
```

Git zeigt ein leeres diff, weil die Version identisch zu der alten Version von uns ist.

git pull

Konfliktfall

- Konfliktauflösung kann Fehlschlagen

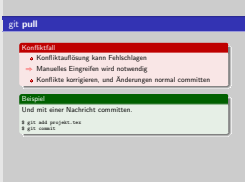
⇒ Manuelles Eingreifen wird notwendig

- Konflikte korrigieren, und Änderungen normal committen

Beispiel

Und mit einer Nachricht committen.

```
$ git add projekt.tex
$ git commit
```

- Arbeit des anderen vernichtet.
- Normalerweise benutzt man Teile von beiden Änderungen, mit evtl. weiteren Änderungen

Literatur

http://git-scm.com Webseite für Git

https://git.wiki.kernel.org Git Wiki

http://www-cs-students.stanford.edu/~blynn/gitmagic/ Buch über Git

Literatur

http://git-scm.com Webseite für Git
https://git.wiki.kernel.org Git Wiki
http://www-cs-students.stanford.edu/~blynn/gitmagic/ Buch über Git

- Viele Tutorials und Bücher im Internet
- Webseite enthält mehr Informationen