Contents

1 HOWTO git

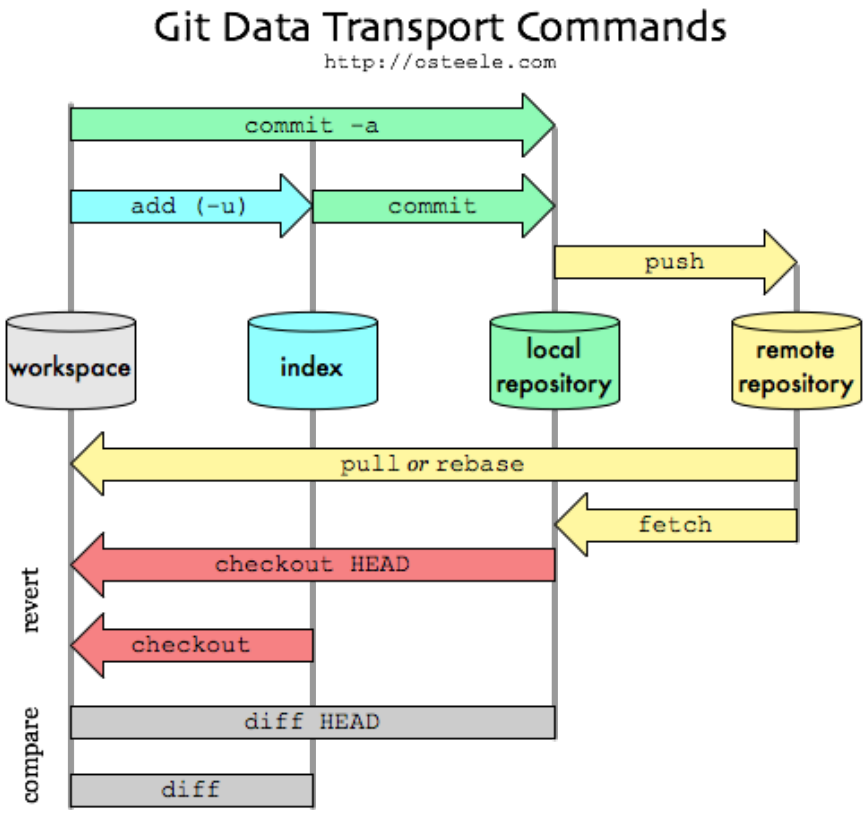
- Anton Hofmann
- `https://git-scm.com/book/de/v1/Git-Grundlagen-%C3%84nderungen-am-Repository-nachv`

2 Intro: 4 Bereiche

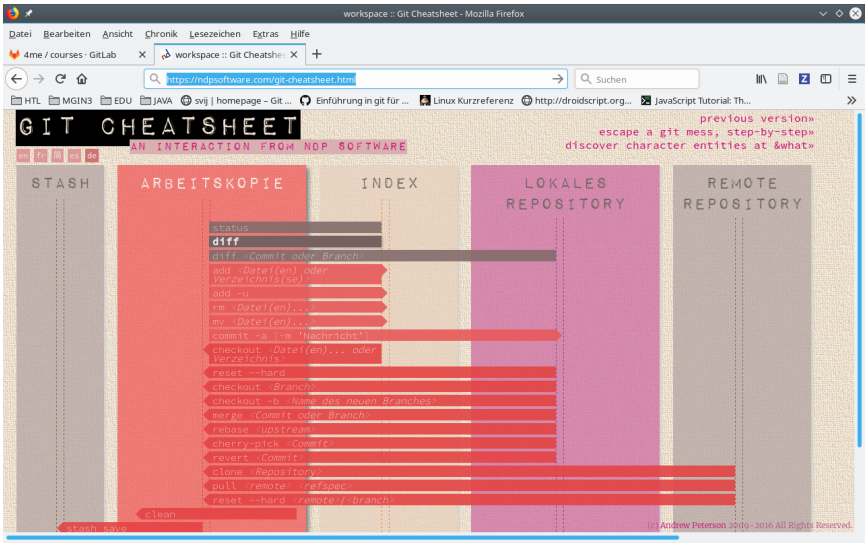
- Arbeitsverzeichnis (working area)
- Staging-Area (index oder stage)
- local repository (committed area)
- remote repository

3 Intro: The big picture

- The Big Picture



- Überblick und Referenz: `https://ndpsoftware.com/git-cheatsheet.html`



4 Übung: Config

```
git config --global user.name "Anton Hofmann"
git config --global user.email "anton.hofmann@htl-salzburg.ac.at"
```

5 Übung: Eigenes Repository erzeugen und clonen

- ein repository auf gitlab.com erzeugen und clonen

```
1. auf gitlab.com ein private repository erzeugen (zB. howto-git)

2. mkdir GITlab
3. cd GITlab
4. git clone https://gitlab.com/<??your-name??>/howto-git.git
5. cd howto-git
```

6 Übung: Ein typischer Arbeitsprozess - README.md

- README.md hinzufügen

```
cd howto-git
nano README.md
git add README.md
```

```
git commit -m "add README"
git push -u origin master
```

7 Übung: Ein typischer Arbeitsprozess - .gitignore

- Was soll nicht ins repository

```
cd howto-git

nano .gitignore
# Keine Executables usw.
*.exe
*.lib
*.dll
*.log
*.class
build/
bin/
temp-*


git add .gitignore
git commit -m".gitignore"
git push -u origin master
```

8 Intro: Branching

- Mehreren Commits einen Namen geben (zB: master, develop, testing, bugfix)
- Erzeugt einen neuen lokalen Branch **testing** und
- wechselt in diesen

```
git branch testing
git checkout testing

oder
git checkout -b testing
```

- Listet alle lokalen Branches im aktuellen Repository auf

```
git branch
```

- Löscht den angegebenen Branch **testing**

```
git branch -d testing
```

9 Übung: Branching

- Der branch **master** existiert bereits
- erstellen Sie auch die folg. Branches:
  - **develop, testing, bugfix**

```
cd howto-git
```

```
git branch develop
git branch testing
git branch bugfix
```

10 Übung: add/commit file[1-3].txt

- Sie aktualisieren ihr lokales Repo mit `git pull`
- Sie arbeiten auf dem branch **master**
- Erstellen Sie nun die Dateien:
  - **file1.txt, file2.txt, file3.txt**
- Erzeugen Sie pro Datei je einen Commit
- laden Sie ihre Änderungen auf gitlab mit `git push`

```
cd howto-git
```

```
git pull origin master
```

```
echo "1" > file1.txt
```

```
git add file1.txt
```

```
git commit -m"file1.txt added"

echo "22" > file2.txt

git add file2.txt
git commit -m"file2.txt added"

echo "333" > file3.txt

git add file3.txt
git commit -m"file3.txt added"

git log

git push origin master
```

- git log liefert zum Beispiel:

```
commit 97a9298c03c1964d8aac4764bf9842746ca0803d (HEAD -> master)
Author: Anton Hofmann <anton.hofmann@htl-salzburg.ac.at>
Date:   Sun Oct 21 18:18:24 2018 +0200

file3.txt added

commit 3d6150f68260f90d4dce55f73b54e0f5948d915b
Author: Anton Hofmann <anton.hofmann@htl-salzburg.ac.at>
Date:   Sun Oct 21 18:18:05 2018 +0200

file2.txt added

commit 034dd5c158af7e8d4c3026c6ad080b9e1c87cb83
Author: Anton Hofmann <anton.hofmann@htl-salzburg.ac.at>
Date:   Sun Oct 21 18:17:23 2018 +0200

file1.txt added

commit af9347d369566d8570f6856fcda5aa6b76e02c33 (origin/master, testing, bugfix)
Author: Anton Hofmann <anton.hofmann@htl-salzburg.ac.at>
Date:   Sun Oct 21 00:02:20 2018 +0200
```

```
erstes commit
```

11 Intro: Merging

- Um die Arbeiten in verschiedenen branches zusammenzufassen.
- Einen ersten Vergleich vor dem eigentlichen `git merge` mit `git diff source_branch target_branch`
- Beispiel:
  - git checkout master
  - git diff testing master
  - git merge testing

12 Übung: Merging ohne Konflikt - testing (add file4.txt)

- im branch testing file4.txt neu hinzufügen und
- merge testing into master

12.1 1. branch testing vorbereiten

1. Verwenden Sie den branch **testing** und
2. fügen Sie die Datei **file4.txt** hinzu

```
cd howto-git

git checkout testing

ls

echo "4444">file4.txt

git add file4.txt
git commit -m"file4.txt added"

git push origin testing
```

12.2 2. merge testing into master

```
git checkout master

ls

git diff testing master

git merge testing

ls
```

- eine mögl. Ausgabe

```
hofmann@u00:/GITlab/howto-git (master>) % git merge testing
Merge made by the 'recursive' strategy.
 file4.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt

hofmann@u00:/GITlab/howto-git (master>) % ls
file1.txt  file2.txt  file3.txt  file4.txt  README.md
hofmann@u00:/GITlab/howto-git (master>) % git status
Auf Branch master
Ihr Branch ist 2 Commits vor 'origin/master'.
  (benutzen Sie "git push", um lokale Commits zu publizieren)

nichts zu committen, Arbeitsverzeichnis unverändert
```

- nun noch

```
git push origin master

git log
```

13 Übung: Merging ohne Konflikt - testing (update file4.txt)

- im branch testing file4.txt ändern und

- merge testing into master

13.1 1. branch testing vorbereiten

- edit file4.txt in branch **testing**

```
cd howto-git

git checkout testing

echo "Hallo, Welt!" >> file4.txt

git add file4.txt
git commit -m"file4.txt update"

git push origin testing
```

13.2 2. merge testing

-

```
git checkout master

git merge testing

cat file4.txt

git push origin master
```

14 Übung: Merging mit Konflikt - master und testing (update file4.txt)

- Branch master **und** branch testing ändern file4.txt und
- erzeugen jeweils ein commit.
- testing: ändere 4444 auf tttt
- master: ändere 4444 auf mmmm

14.1 1. branch testing und master ändern file4.txt

```
git checkout testing

nano file4.txt

git add file4.txt
git commit -m "file4.txt update again"


git checkout master

nano file4.txt

git add file4.txt
git commit -m "file4.txt update again"
```

14.2 2. merge testing - Es wird einen Konflikt geben

```
git checkout master
git merge testing
```

- Hier ist der Konflikt

```
hofmann@u00:/GITlab/howto-git (master>) % git merge testing
automatischer Merge von file4.txt
KONFLIKT (Inhalt): Merge-Konflikt in file4.txt
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann d
```

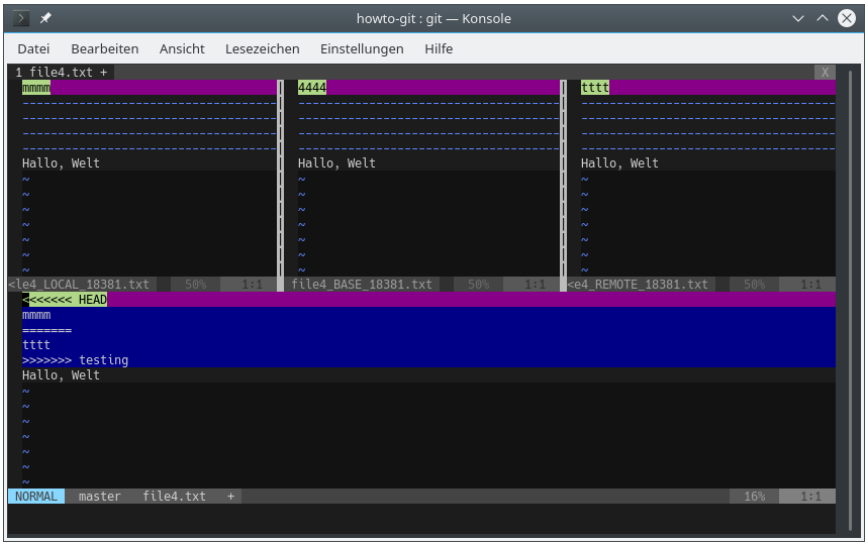
14.3 3. merge mit mergetool

1. config mergetool

```
git config merge.tool vimdiff
git config merge.conflictstyle diff3
git config mergetool.prompt false
```

1. starte mergetool

```
git mergetool
```



- BASE::die Version vor dem letzten commit
- LOCAL::die Version des aktuellen branch
- REMOTE::die Version des zu mergenden branch
- MERGED::die gewollte Version
- Wählen Sie eine Version durch den Befehl

```
:diffget LOCAL
oder
:diffget BASE
oder
:diffget REMOTE

und dann editiere MERGED

und dann verlassen Sie das Tool mit
:wqa
```

1. add und commit

```
git add file4.txt
git commit -m"file4.txt merged with conflict"

git push origin master
```

15 Intro: Änderungen synchronisieren (lokal->remote)

- Austauschen der Repository-Historie
- Pusht alle lokalen Commits zum remote (unter origin bekannt) branch (hier master)

```
$ git push origin master
```

- Pullt die Historie vom externen Repository und integriert die Änderungen

```
git pull origin master
```

- Lädt die gesamte Historie eines externen Repositories herunter

```
git fetch origin
```

16 Intro: Historie und Logging

- Listet die Versionshistorie für den aktuellen Branch auf

```
git log
```

- Listet die Versionshistorie für die aktuelle Datei auf, inklusive Umbenennungen

```
git log --follow filename.txt
```

- Zeigt die inhaltlichen Unterschiede zwischen zwei Branches

```
git diff master testing
```

- Gibt die Änderungen an Inhalt und Metadaten durch den angegebenen Commit aus

```
git show fb56342
```

17 Info: Arbeitsverzeichnis und die Staging-Area im Detail

- Listet alle zum Commit bereiten neuen oder geänderten Dateien auf

```
git status
```

- zum **Hinzufügen** in die Staging-Area aufnehmen

```
git add filename.txt
```

- zum **Umbennen** im Arbeitsverzeichnis und in der Staging-Area

```
git mv filename.txt filename-renamed.txt
```

- zum **Löschen** im Arbeitsverzeichnis und in der Staging-Area.

```
git rm -f filename.txt
```

- zum **Verwerfen** von (falschen) Änderungen im Arbeitsverzeichnis

```
git checkout -- filename.txt
```

- zum **Entfernen aus der Staging-Area.**
- filename.txt **bleibt** im Arbeitsverzeichnis

```
git reset filename.txt
oder
git rm --cached filename.txt
```

- Zeigt die Unterschiede zwischen dem Arbeitsverzeichnis und der Staging-Area

```
git diff --staged
```

- Gibt alle derzeit indizierten Dateien permanent in die Versionshistorie auf

```
git commit -m"Ein erster Versuch der Versionierung"
```

18 Info: Commits rückgängig machen

- Fehler beseitigen und die Historie bereinigen
- Macht alle Commits nach dem commit fb56342 rückgängig, erhält die Änderungen aber lokal

```
git reset fb56342
```

- Verwirft die Historie und Änderungen seit dem angegebenen Commit

```
git reset --hard fb56342
```

- wenn die lokalen Änderungen komplett entfernen werden sollten,
- holt man den letzten Stand vom entfernten Repository mit folgenden Befehlen:

```
git fetch origin
git reset --hard origin/master
```

19 Info: Tagging

- Es wird empfohlen, für Software Releasestags zu verwenden.
- 1b2e1d63ff steht für die ersten 10 Zeichen der Commit-Id.
- Es können auch weniger Zeichen sein (aber eindeutig)

```
git tag 1.0.0 1b2e1d63ff
```

```
git log
```

20 Info: Links

- Visuelles: (L)
  - `http://ndpsoftware.com/git-cheatsheet.html`
- Git (Beispiele mit bootstrap)
  - `https://svij.org/blog/2014/10/25/git-fur-einsteiger-teil-1/`
  - `https://svij.org/blog/2014/11/01/git-fur-einsteiger-teil-2/`

- Ein schneller Überblick:
  - http://www.nullpointer.at/2011/10/16/howto-git-commands-ein-uberblick/
  - https://rogerdudler.github.io/git-guide/index.de.html
- Git-Book:
  - https://git-scm.com/book/de/v1/Git-Grundlagen
  - http://gitbu.ch/pr01.html

21 Info: Weitere Themen

21.1 Aufschieben und Wiederherstellen unvollständiger Änderungen

- Speichert temporär alle getrackten Dateien mit Änderungen

```
git stash
```

- Stellt die zuletzt zwischengespeicherten Dateien wieder her

```
git stash pop
```

- Listet alle zwischengespeicherten Änderungen auf

```
git stash list
```

- Verwirft die zuletzt zwischengespeicherten Änderungen

```
git stash drop
```

21.2 Beispiel: Branching and Merging

- https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
- Let's go through a simple example of branching and merging.
- You'll follow these steps:
  - Do some work on a website.
  - Create a branch for a new story you're working on.
  - Do some work in that branch.

- At this stage, you'll receive a call that another issue is critical and you need a hotfix.
- You'll do the following:
  - Switch to your production branch.
  - Create a branch to add the hotfix.
  - After it's tested, merge the hotfix branch, and push to production.
  - Switch back to your original story and continue working.