

# GIT-Training

## Agenda

### 1. Geschichte / Grundlagen

- [GIT Pdf](#)

### 2. Workflows

- [git workflows](#)

### 3. Commands (with tips & tricks)

- [git alias](#)
- [git add + Tipps & Tricks](#)
- [git commit](#)
- [git log](#)
- [git config](#)
- [git show](#)
- [Needed commands for starters](#)
- [git branch](#)
- [git checkout](#)
- [git merge](#)
- [git tag](#)
- [git rm \(Dateien löschen aus git\)](#)

### 4. Erweiterte Commands

- [git reflog](#)
- [git reset - Back in Time](#)

### 5. Tipps & tricks

- [Beautified log](#)
- [Change already committed files and message](#)
- [Best practice - Delete origin, tracking and local branch after pull request/merge request](#)
- [Einzelne Datei auschecken](#)
- [Always rebase on pull - setting](#)
- [Arbeit mit submodules](#)
- [Integration von Änderungen \(commits, einzelne Dateien\) aus anderen commits in den Master](#)
- [Fix conflict you have in merge-request \(gitlab\)](#)
- [SETUP.sql zu setup.sql in Windows \(Groß- und Kleinschreibung\)](#)
- [Force specific commit message](#)
- [Alle Dateien, die sich geändert haben anzeigen z.B. heute](#)

### 6. Tipps & Tricks (Mergen)

- [No automerging - please](#)

### 7. Exercises

- [merge feature/4712 - conflict](#)
- [merge request with bitbucket](#)
- [merge request bitbucket with conflict](#)
- [Exercise with cherry-picking](#)

- [Gruppenarbeit-bitbucket-ohne-konflikt](#)
- [Gruppenarbeit bitbucket mit Konflikt](#)

## 8. Snippets

- [publish lokal repo to server - bitbucket](#)
- [failure-on-push-fix](#)
- [failure-on-push-with-conflict](#)

## 9. Extras

- [Best practices](#)
- [Using a mergetool to solve conflicts](#)
- [Overview GIT-Servers](#)
- [4 goldene Regeln](#)

## 10. Help

- [Help from commandline](#)

## 11. subtrees / submodules

- [subtrees](#)
- [submodules](#)

## 12. Authentication

- [Work with different credentials](#)

## 13. Documentation

- [GIT Pdf](#)
- [GIT Book EN](#)
- [GIT Book DE](#)
- [GIT Book - submodules](#)
- [GIT Guis](#)
- [Third Party Tools](#)
- [Specification Conventional Commits](#)
- <https://www.innog.com/de/talks/2019/05/commit-message-101/>
- <https://github.com/GitAlias/gitalias/blob/main/gitalias.txt>
- <https://education.github.com/git-cheat-sheet-education.pdf>

## 14. Integrations

- <https://docs.gitlab.com/ee/integration/jira/>

## 15. GUIs

- [git extensions gui](#)
- [gui uebersicht](#)

# Backlog

## 1. Installation

- [GIT auf Ubuntu/Debian installieren](#)
- [GIT unter Windows installieren](#)

## Geschichte / Grundlagen

### GIT Pdf

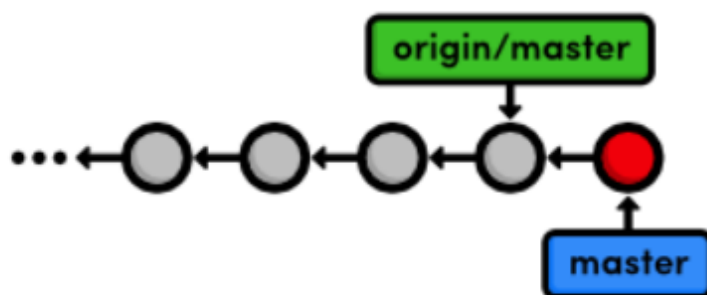
- <http://schulung.t3isp.de/documents/pdfs/git/git-training.pdf>

## Workflows

### git workflows

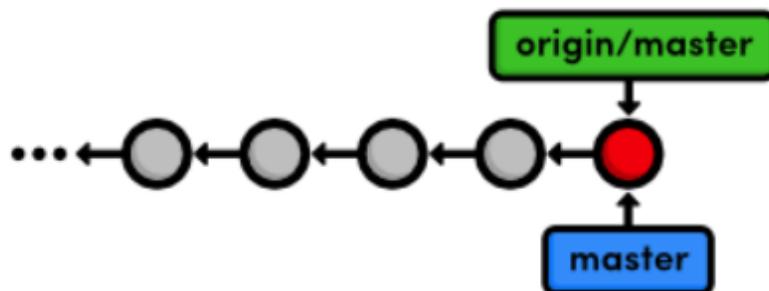
#### Centralized Workflow

#### Our Repository, Before Pushing

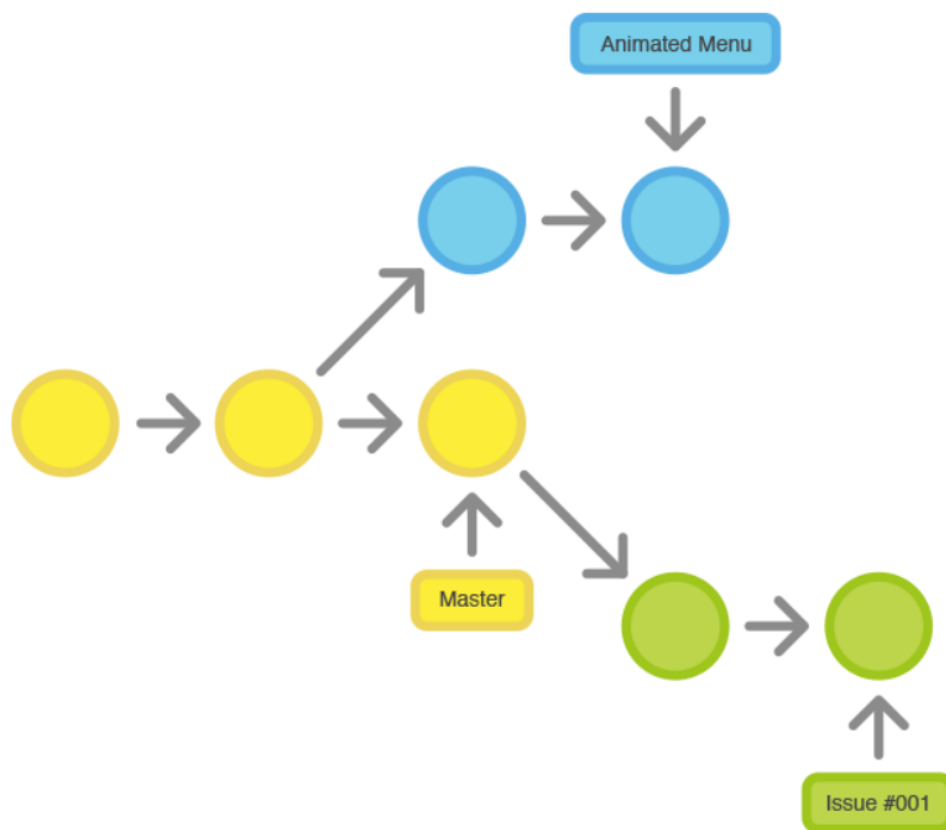


---

#### Our Repository, After Pushing



#### Feature Workflow



## Gitflow workflow



## Commands (with tips & tricks)

### git alias

```
git config --global alias.sl '!git log --oneline -2 && git status'
```

### git add + Tipps & Tricks

#### Trick with -A

```
## only adds from the folder you are in recursively
## but not above (you might miss some files, when you are in a subfolder
git add .

### Fix -A
## adds everything no matter in which folder you are in your project
git add -A
```

### git commit

#### commit with multiple lines on commandline (without editor)

```
git commit -am "New entry in todo.txt

* nonsense commit-message because of missing text-expertise"
## enter on last line
```

#### Change last commit-message (description)

```
git commit --amend
## now you can change the description, but you will get a new commit-id
```

### git log

#### Show last x entries

```
##
## git log -x
## Example: show last 2 entries
git log -2
```

#### Show all branches

```
git log --all
## oder wenn alias alias.lg besteht:
## git lg --all
```

#### Show first log entry

```
## Step 1 - log needs to only show one line per commit
git log --oneline --reverse

## Step 2: combine with head
git log --oneline --reverse | head -1
```

## Multiple commands with an alias

```
git config --global alias.sl '!git log --oneline -2 && git status'
```

## git config

### How to delete an entry from config

```
## Important: Find exact level, where it was added --global, --system, --local
## test before
## should contain this entry
git config --global --list

git config --unset --global alias.log
```

## git show

### Show information about an object e.g. commit

```
git show <commit-ish>
## example with commit-id
git show 342a
```

## Needed commands for starters

```
git add -A
git status
git log // git log -4 // or beautified version if setup as alias git lg
git commit -am "commit message" // "commit message" can be freely chosen
## for more merge conflict resolution use only
git commit # to not change commit - message: must be message with merge
## the first time
git push -u origin master
## after that
git push
git pull
```

## git branch

### Create branch based on commit (also past commit)

```
git branch lookaround 5f10ca
```

## Delete unmerged branch

```
git branch -d branchname # does not work in this case
git branch -D branchname # <- is the solution
```

## Delete remote tracking branch

```
git branch -d -r origin/feature/501
```

## git checkout

### Checkout (change to) existing branch

```
git checkout feature/4711
```

### Checkout and create branch

```
## Only possible once
git checkout -b feature/4712
```

### File aus einem Commit holen (oder HEAD)

```
git checkout HEAD -- todo.txt
```

## git merge

### Merge without conflict with fast-forward

```
## Disadvantage: No proper history, because only one branch visible in log
## after fast-forward - merge

## Important that no changes are in master right before merging
git checkout master
git merge feature/4711
```

### Merge (3-way) also on none-conflict (no conflicts present)

```
git merge --no-ff feature/4711
```

## git tag

### Creating tags, Working with tags

```
## test
## set tag on current commit -> HEAD of branch
git tag -a v1.0 -m "my message for tag"
## publish
```

```
git push --tags

## set on specific commit
git tag -a v0.1 -m "Initial Release" a23c

## checkout files of a specific tag
git checkout v0.1
## or
git checkout tags/v0.1
```

## git delete tag

```
## Tag local löschen und danach online löschen
git tag -d test.tag
git push --delete origin test.tag

## Tag online löschen und danach lokal
## Schritt 1: Über das interface (web) löschen
## Schritt 2: aktualisieren
git fetch --prune --prune-tags
```

## Misc

```
## Fetch new tags from online
git fetch --tags

## Update master branch (rebase) and fetch all tags in addition from online
git checkout master
git pull --rebase --tags
```

## git rm (Dateien löschen aus git)

### Datei komplett löschen (Workspace und Repo)

```
git rm dateiname
```

### Datei nur aus Repo und Index löschen

```
git rm --cached dateiname
```

## Erweiterte Commands

### git reflog

#### command

- show everything you (last 30 days), also stuff that is not visible in branch anymore

#### Example



```
git reflog
```

## when many entries a pager like less (aka man less) will be used

```
## you can get out of the page with pressing the key 'q'
```

## git reset - Back in Time

### Why ?

- Back in time -> reset
- e.g. git reset --hard e2d5
- attention: only use it, when changes are not published (remotely) yet.
- → It is your command, IN CASE your are telling yourself, omg, what's that, what did i do here, let me undo that

### Example

```
git reset --hard 2343
```

## Tipps & tricks

### Beautified log

### Walkthrough

```
git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset \
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'"
```

## PRETTY FORMATS

- all documented in git help log (section PRETTY FORMATS)
- <https://git-scm.com/docs/git-log>

## Change already committed files and message

```
## Walkthrough
touch newfile.txt
git add .
git commit -am "new file added"

## Uups forgotten README
touch README
git add .
git commit --amend # README will be in same commit as newfile.txt
## + you can also changed the commit message
```

## Best practice - Delete origin,tracking and local branch after pull request/merge request

```
## After a succesful merge or pull request und gitlab / github
## Follow these steps for a succesful cleanup
```

```
## 1. Delete feature branch in web interface (e.g. gitlab / github)
## e.g. feature/4811

## 2. Locally on your system prune the remote tracking branch
git fetch --prune

## 3. Switch to master or main (depending on what you master branch is)
git checkout master

## 4. Delete local branch
git branch -d feature/4811
```

## Einzelne Datei auschecken

### aus anderem Commit

```
## aus commit 11ed

git checkout 11ed -- todo.txt
## unterverzeichnis
git checkout 11ed -- tmp/test.txt
```

## ...und direkt umbenennen

```
## datei todo.txt aus 11ae -> Inhalt anzeigen und direkt neue datei umleiten
git show 11ae:todo.txt > todoneu.txt

## ein commit vorher
git show 11ae^:todo.txt > todoneu.txt
```

## Always rebase on pull - setting

```
git config branch.master.rebase true
```

## Arbeit mit submodules

### Add submodule

```
git submodule add https://github.com/jmetzger/training-git.git
```

## Clone repo with submodules (Important !)

```
git clone --recurse-submodules https://gitlab.com/dummyhoney/jochen111.git training-
subtest
```

## Updaten des submodules

```
git submodule update --remote training-git
git commit -am "new version"
```

## Best practice

```
clone repo use for submodule seperately
(in seperate folder)
if you want to change it
```

## Updating commands for updating subfolder

```
git submodule update --remote
## use other branch from submodule then master
git config -f .gitmodules submodule.DbConnector.branch stable
```

## Get rid of submodule

```
rm -fR training-git/
git rm .gitmodules
git rm training-git
git status
git commit -am "removed submodules"
```

## Ref.

- <https://git-scm.com/book/de/v2/Git-Tools-Submodule>

## Integration von Änderungen (commits, einzelne Dateien) aus anderen commits in den Master

### Walkthrough

```
## 1. Schritt - erstellen integrationsbranch von dev/staging branch
git checkout -b integrate/1

## Möglichkeit 1: cherry-pick - komplette commit inkl. aller Änderungen mit reinnehmen
## Hier wird gemerged: Gemerged
## Evtl. Konflikt, den muss ich dann lösen
git cherry-pick c5906c0

## Möglichkeit 2: Einzelne files aus commit: Achtung, wenn im Work-Directory
## bereits vorhanden überschrieben
## commit wird bereits durchgeführt
git checkout ddb0 -- armin3.txt

## Möglichkeit 3: cherry-pick ohne commit
git cherry-pick -n 4497
git status
## alle files rausnehmen, die wir nicht haben möchten, wie folgt.
git restore --staged agenda.txt
```

```
## Achtung, jetzt sind diese so im Working Directory als unstaged
## d.h. die alte Version aus dem letzten Commit holen
git checkout HEAD -- agenda.txt

## 3. Schritt
## Änderungen commiten
git commit -am "Revised version"

## 4. Nach online pushed
git push -u origin integrate/1

## 5. Merge request in gitlab: integrate/1 -> master
## und dann mergen online
```

## Fix conflict you have in merge-request (gitlab)

### Walkthrough

```
## create feature-branch and worked on it
git checkout -b feautre/4711
## ... changes
git add .; git commit -am "new feature"
## pushed branch online
git push -u origin feature/4711
## then created merge online
## feature/4711 --> master

##### TaDa - It was NOT possible to merge because of conflict
## unfortunately advice on gitlab/bitbucket is not worth the dime

## locally, update you feature-branch like so
## NO git pull --rebase please, otherwise, you have to redo you merge_request
afterwards
## get changes from master
git pull origin master

## fix conflicts
git add .
git commit

## push new version of feature - branch online
git push

## now you can merge in the merge-request interface on gitlab
```

## SETUP.sql zu setup.sql in Windows (Groß- und Kleinschreibung)

### Problem

- Windows erkennt in git keine Änderung der Groß- und Kleinschreibung
- Workaround: git rm --cached; git commit -am

## Walkthrough

```
touch SETUP.sql
git add .; git commit -am "SETUP neu"

## Ups, verschrieben ! Was jetzt ?
git rm --cached SETUP.sql # Datei wird aus git rausgenommen
git commit -am "und dingfest machen"
## Beweis
git show HEAD # letztes commit mit Änderungen anzeigen

## Jetzt auf ein Neues
## oder im Explorer
mv SETUP.sql setup.sql
git add .; git commit -am "setup.sql neu"
git show HEAD
```

## Force specific commit message

### Basics

- Done on Server-Side
- Specific to server - Software (like github/gitlab)

### Example - pre-receive-hook

- <https://git-scm.com/book/en/v2/Customizing-Git-An-Example-Git-Enforced-Policy>

### Ref:

- [https://docs.gitlab.com/ee/user/project/repository/push\\_rules.html](https://docs.gitlab.com/ee/user/project/repository/push_rules.html) (not free)
- [https://docs.gitlab.com/ee/administration/server\\_hooks.html](https://docs.gitlab.com/ee/administration/server_hooks.html)

## Alle Dateien, die sich geändert haben anzeigen z.B. heute

### Files

```
git log --after="2015-11-05T16:36:00-02:00" --before="2022-09-28" --pretty=format:"" --name-only | sort -u
```

### Mit loop

```
for i in $(git log --after="2022-09-26" --before="2022-09-27" --pretty=format:"" --name-only | sort -u); do git log -- $i; done
```

## Änderungen einer datei

```
git log --after="2022-09-26" --before="2022-09-27" --pretty=format:"" --follow -p --todo.txt
```

## Tipps & Tricks (Mergen)

### No automerging - please

## Mergen ohne commit, commit selbst nach Überprüfung

```
git merge --no-commit --no-ff <local-branch>
## schritt 2:
## Entweder. Vergleichen mit diff
## d.h. Index wird verglichen mit letzten Commit
git diff HEAD
## Oder schön mit difftool (wenn konfiguriert)
git difftool HEAD
```

## Exercises

### merge feature/4712 - conflict

#### Exercise

```
1. You are in master-branch
2. Checkout new branch feature/4723
3. Change line1 in todo.txt
4. git add -A; git commit -am "feature/4723 done"
5. Change to master
6. Change line1 in todo.txt
7. git add -A; git commit -am "change line1 in todo.txt in master"
8. git merge feature/4723
```

### merge request with bitbucket

```
## Local
git checkout -b feature/4822
ls -la
touch f1.txt
git add .
git commit -am "f1.txt"
touch f2.txt
git add .
git commit -am "f2.txt"
git push -u origin feature/4822
```

### Online bitbucket / gitlab

```
## create merge request
## and merge
```

### Delete branch online after merge

- eventually done automatically when checkbox was set
- or: delete from branches menu

### Cleanup locally

```
git fetch --prune
git checkout master
git branch -D feature/4822
git pull --rebase
```

### merge request bitbucket with conflict

```
## Local
git checkout -b feature/5021
## ändern zeile1 in todo.txt
notepad todo.txt

git add .
git commit -am "aenderung todo.txt"
git push -u origin feature/5021
```

### Online Änderung im master -> todo.txt -> Zeile 1

Änderung über web-Oberfläche in bitbucket -> source

### Online bitbucket / gitlab

```
## create merge request
## and merge --> conflict
```

### Auflösen des Konflikts (im Branch feature/5021)

```
git pull origin master
## lösen den conflict

git status

## modifizieren die todo.txt
notepad todo.txt

git add todo.txt
## Konflikt gelöst
git commit

## der branche wird nochmal hochgeschoben
git push
```

### Merge durchführen

```
## Wieder in den Pull-Request rein und den Merge durchführen
```

### Delete branch online after merge

- eventually done automatically when checkbox was set

- or: delete from branches menu

## Cleanup locally

```
git fetch --prune
git checkout master
git branch -D feature/5021
git pull --rebase
```

## Exercise with cherry-picking

### Walkthrough

```
1. Neuen Branch feature/5050 erstellen
2. 3 Änderungen wie folgt:
  a. todo.txt Zeile1 + add -A + commit
  b. todo.txt Zeile2 + add -A + commit
  c. todo.txt Zeile3 + add -A + commit
3. Wechsel in den master
---
4. commit von 2b. notieren
5. branch löschen
6. Cherry-picken von commit aus 2b
```

## Gruppenarbeit-bitbucket-ohne-konflikt

### Phase 1

Jeder in der Gruppe erstellt lokal ein Feature

```
## Local
## git checkout -b feature/<euer-vorname>
## e.g.
git checkout -b feature/jochen1
ls -la
touch jochen1.txt
git add -A
git commit -am "jochen1.txt"
git push -u origin feature/jochen1
```

### Online bitbucket / gitlab

```
## create merge request
```

### Phase 2

### Online bitbucken - strukturiert mergen

```
## and mergen strukturiert nacheinander
```



## Delete branch online after merge

- eventually done automatically when checkbox was set
- or: delete from branches menu

## Cleanup locally

```
git fetch --prune
git checkout master
git branch -D feature/<euer-vorname>
git pull --rebase
```

## Gruppenarbeit bitbucket mit Konflikt

### Phase 1

#### Jeder in der Gruppe erstellt lokal ein Feature

```
## Local
## git checkout -b feature/<euer-vorname>
## e.g.
git checkout -b feature/jochen2
## Zeile 1 todo.txt
notepad todo.txt
git add -A
git commit -am "todo.txt"
git push -u origin feature/jochen2
```

#### Online bitbucket / gitlab

```
## create merge request
```

### Phase 2

#### Online bitbucket - strukturiert mergen

```
## and mergen strukturiert nacheinander
## conflict
```

#### Jetzt conflict lokal lösen

```
## in unserem feature branch
git pull origin master

## conflict auflösen
notepad todo.txt
## entscheiden für codeblock

## ändern kenntlich machen
git status
git add todo.txt
```

```
## merge ist fertig
git commit

git push
```

## Online mergen

```
### Jetzt dürfte kein Konflikt mehr da sein
```

## Delete branch online after merge

- eventually done automatically when checkbox was set
- or: delete from branches menu

## Cleanup locally

```
git fetch --prune
git checkout master
git branch -D feature/<euer-vorname>
git pull --rebase
```

# Snippets

## publish lokal repo to server - bitbucket

```
# Step 1: Create repo on server without README and .gitignore /set both to NO when
creating

# Step 2: on commandline locally
cd /path/to/repo
git remote add origin https://erding2017@bitbucket.org/erding2017/git-remote-
jochen.git
git push -u origin master

# Step 3: for further commits
echo "test" > testdatei
git add .
git commit -am "added testdatei"
git push
```

## failure-on-push-fix

```
## Step 1: push produces error
## you have done git push -u origin master the last to setup remote tracking branch by
option -u
git push
Password for 'https://erding2017@bitbucket.org':
To https://bitbucket.org/erding2017/git-remote-jochen.git
! [rejected]        master -> master (fetch first)
```

```

error: failed to push some refs to 'https://erding2017@bitbucket.org/erding2017/git-remote-jochen.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
## Step 2: Integrate changes from online
git pull
## Step 2a: Editor opens and you need to save and ext (without changing anything)

## Step 3: re-push
git push

```

## failure-on-push-with-conflict

## Failure push

```

## Step 1: push produces error
## you have done git push -u origin master the last to setup remote tracking branch by
option -u
git push
Password for 'https://erding2017@bitbucket.org':
To https://bitbucket.org/erding2017/git-remote-jochen.git
 ! [rejected]          master -> master (fetch first)
....
## Step 2: Integrate changes from online
git pull

## Step 3: Solve conflict
Auto-merging agenda.txt
CONFLICT (content): Merge conflict in agenda.txt
Automatic merge failed; fix conflicts and then commit the result.
kurs@ubuntu-tr01:~/training$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
    (use "git pull" to merge the remote branch into yours)

## Step 3a: Open file agenda.txt
## Decide for which version
## - remove all <<<<< and ===== and >>>>>>>>> - lines

## Step 3b: then: save + exit from editor

## Step 3c: mark resolution
git status
git add todo.txt

## Step 3d:

```

```
git status
## as written there
git commit

## Step 4: re-push
git push
```

## recipe

```
git push # failure
git pull
git add todo.txt
git commit
git push
```

## Extras

### Best practices

- Delete branches, not needed anymore
- `git merge --no-ff ->` for merging local branches (to get a good history from local)
- from online: `git pull --rebase //` clean history from online, not to many branches
- nur auf einem Arbeiten mit max. 2 Teilnehmern, wenn mehr feature-branch

## Teil 2:

- Be careful with git commands that change history.
  - never change commits, that have already been pushed
- Choose workflow wisely
- Avoid `git push -f` in any case // should not be possible
- Disable possibility to push -f for branch or event repo

## Using a mergetool to solve conflicts

### Meld (Windows) - Install

- <https://meldmerge.org/>

### Find out if mergetool meld is available

```
## Important: close and reopen git bash before doing that
## you can try to see, if meld can be executed by simply typing "meld"

git mergetool --tool-help
```

### Configure, when it is found by mergetool --tool-help

```
## you have to be in a git project
git config --global merge.tool meld
git config --global diff.tool meld
```

```
git config --global mergetool.keepBackup false
git config --list
```

## If not found bei mergetool --tool-help :: Configuration in Git for Windows (git bash)

```
## you have to be in a git project
git config --global merge.tool meld
git config --global diff.tool meld
## Should be on Windows 10
git config --global mergetool.meld.path
"/c/Users/Admin/AppData/Local/Programs/Meld/Meld.exe"
## sometimes here
git config --global mergetool.meld.path "/c/Program Files/Meld/Meld.exe"
## do not create an .orig - file before merge
git config --global mergetool.keepBackup false
```

## How to use it

```
## when you have conflict you can open the mergetool (graphical tool with )
git mergetool
```

## Overview GIT-Servers

### Builtin with git-installation

#### Simple GIT-Server

```
## included in installation with git
Cons: Can do nearly nothing (only pushing and pulling)

* no graphical interface
* no multi-user support
* no additional features (like bugtracking / milestones a.s.o)
```

#### Web-Interface (also from git installation)

```
Cons: Mo multi-user interaction
```

### Comfortable Git-Server

#### gitea / codeberg

- OpenSource
- minimum feature
- not integrated with other software

#### gitlab

##### General

- On premise / cloud

##### Pros

- Devops - Server (Integration)

- Tools für Devops
- Integration von CI/CD
  - Favourite von Jochen (in opposite github actions)
- kleine Teams können on premise kostenlos starten
- Im Rahmen von DevOps auch automatische Integration von Scannen von Software drin.

## **bitbucket**

### **Overview**

- Software Company Atlassian.
- Problematic license policy
- Cloud-Based (SaaS) - ich miete - subscription
- On Premise (Installation im Firmennetz)
  - aber abgekündigt
- On Premise für grosse Unternehmen - sehr teuer

### **Pros**

- Integration with other software products (confluence - wiki, jira - ticket system)
- webhooks (url aufgerufen wird dich ich festlege mit einem payload)

### **Cons**

- No CI/CD directly within bitbucket

## **github**

### **Overview**

- Bought by microsoft

### **Pros**

- on premise git gut möglich (github enterprise)
- Editor sehr gut im Web-Interface

### **Cons**

- Menüführung von github nicht so intuitiv für Jochen
- github actions (CI/CD) zu kompliziert (Lernkurve größer als bei gitlab ci/cd)

## **Azure Devops**

### **Overview**

- Repos are use from github under the hood

### **Con**

- Lernkurve höher als bei github, gitlab, bitbucket

### **Pros**

- Sicherheitsfeatures höher
- Integration mit VisualStudio
- Kostenvorteile durch Lizenz Visual Studio Pro

## **AWS Code Commit**

### **Overview**

- Innerhalb der Amazon AWS Familie

### **Pros**

- Integration von AWS

### **Cons**

- Etwas ungünstige Positionierung des Interface (wo finde ich das überhaupt)
- Benennung: AWS Console -> Web Interface
- Sehr kleines FeatureSet (z.B. GIT LFS möglich)

- keinen Forken möglich

## 4 goldene Regeln

```
* Niemals einen push --force machen
  (nur in Abstimmung mit dem gesamten Team)
* kein reset vor bereits veröffentlichte commits
* git commit --amend nur wenn commit noch nicht veröffentlicht (push auf server)
* rebase nur wenn branch / commit noch nicht veröffentlicht
```

## Help

### Help from commandline

### On Windows

```
## on git bash enter
git help <command>
## e.g.
git help log

## --> a webpage will open with content
```

## subtrees / submodules

### subtrees

### Prerequisites - Existing local repo

```
## in der bash
cd ..
cp -a training training-neu
cd training-neu
```

### Walkthrough

```
## -f is needed because commits are different from main project
git remote add -f training-git https://github.com/jmetzger/training-git.git
git status
git subtree add --prefix training-git training-git main --squash
```

### Updating

```
git fetch training-git main
git subtree pull --prefix training-git training-git main --squash
```

### Push

```
git subtree push --prefix=training-git training-git main
```

## Ref.

- <https://www.atlassian.com/git/tutorials/git-subtree>

## submodules

### Add submodule

```
git submodule add https://github.com/jmetzger/training-git.git
```

### Clone repo with submodules (Important !)

```
git clone --recurse-submodules https://gitlab.com/dummyhoney/jochen111.git training-  
subtest
```

### Updaten des submodules

```
git submodule update --remote training-git  
git commit -am "new version"
```

### Best practice

```
clone repo use for submodule seperately  
(in seperate folder)  
if you want to change it
```

### Updating commands for updating subfolder

```
git submodule update --remote  
## use other branch from submodule then master  
git config -f .gitmodules submodule.DbConnector.branch stable
```

### Get rid of submodule

```
rm -fR training-git/  
git rm .gitmodules  
git rm training-git  
git status  
git commit -am "removed submodules"
```

## Ref.

- <https://git-scm.com/book/de/v2/Git-Tools-Submodule>

## Authentication

### Work with different credentials

## Ref:

<https://de.linkedin.com/pulse/mehrere-gitlabgithub-accounts-bzw-ssh-keys-zum-host-mit-mindermann>



## Documentation

### GIT Pdf

- <http://schulung.t3isp.de/documents/pdfs/git/git-training.pdf>

### GIT Book EN

- <https://git-scm.com/book/en/v2>

### GIT Book DE

- <https://git-scm.com/book/de/v2>

### GIT Book - submodules

- <https://git-scm.com/book/de/v2/Git-Tools-Submodule>

### GIT Guis

- <https://git-scm.com/downloads/guis/>

## Third Party Tools

### Continuous Integration / Continuous Deployment (CI/CD)

```
## Test often / Test automated (CI)

* Jenkins
* Github Actions
* Git Webhooks

## Publish new versions frequently (CD)

* Jenkins
* Github Action
* Git Webhooks
```

### Specification Conventional Commits

- <https://www.conventionalcommits.org/en/v1.0.0/>

## Integrations

### GUIs

#### git extensions gui

#### Installation

- <http://gitextensions.github.io/>

#### gui uebersicht

- <https://git-scm.com/downloads/guis>

## Installation

## **GIT auf Ubuntu/Debian installieren**

### **Installation**

```
sudo apt update  
sudo apt install git
```

### **Language to english please !!**

```
sudo update-locale LANG=en_US.UTF-8  
su - kurs  
  
## back to german  
  
sudo update-locale LANG=de_DE.UTF-8  
su - kurs  
  
## Reference:  
https://www.thomas-krenn.com/de/wiki/Locales\_unter\_Ubuntu\_konfigurieren  
  
## update-locale does a change in  
$ cat /etc/default/locale  
LANG=en_US.UTF-8
```

### **GIT unter Windows installieren**

- <https://git-scm.com/download/win>