GIT-Training (tortoise git)

Agenda

1. Geschichte / Grundlagen
    - GIT Pdf
2. Daily work (best practices)
    - Best practices für daily work mit feature-branches
3. Installation / Nutzung (tortoisegit)
    - Installation tortoisegit
    - Neues Repo einrichten
    - tortoisemerge für git bash einrichten
    - Fehlende Icons anzeigen - FIX
4. Commands git (with tipps & 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)
5. Erweiterte Commands
    - git reflog
    - git reset - Back in Time
6. 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
    - conflict you have in merge-request (gitlab)
    - SETUP.sql zu setup.sql in Windows (Groß- und Kleinschreibung)
    - Force specfic commit message
    - Alle Dateien, die sich geändert haben anzeigen z.B. heute
7. Tipps & Tricks (Mergen)
    - No automerging - please
8. Tipps & Tricks (Tags)
    - Delete Tags being deleted online also locally
9. Tipps & Tricks (.gitignore)

- Exercise: fix already committed files for .gitignore

10. Editoren / Editoren konfigurieren

    - Editoren RapiOS bzw. Raspian
    - Notepad++

11. Exercises

    - merge feature/4712 - conflict
    - merge request with bitbucket
    - merge request bitbucket with conflict - KOCHREZEPT
    - Exercise with cherry-picking
    - Gruppenarbeit-bitbucket-ohne-konflikt
    - Gruppenarbeit bitbucket mit Konflikt

12. Snippets

    - publish lokal repo to server - bitbucket
    - failure-on-push-fix
    - failure-on-push-with-conflict

13. Mergetool

    - Using a mergetool to solve conflicts
    - Using tortoisegit as a mergetool to solve conflicts

14. Extras

    - Best practices
    - Overview GIT-Servers
    - 4 goldene Regeln

15. Help

    - Help from commandline

16. subtrees / submodules

    - substrees
    - submodules

17. submodules in tortoisegit

    - submodules in tortoisegit

18. Authentication

    - Work with different credentials

19. Tortoise - Documentation

    - https://tortoisegit.org/docs/tortoisegit

20. Remote in Teams arbeiten

    - Workflows
    - Branches and Onlinebranches
    - Fix conflict in pull-request - gitlab

21. Documentation - .gitignore

    - List for different programming lanuages

22. Documentation - permissions

    - Permissions

23. Documentation

    - GIT Pdf
    - GIT Book EN
    - GIT Book DE
    - GIT Book - submodules
    - GIT Guis
    - Third Party Tools
    - Specification Conventional Commits
    - https://www.innoq.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

Backlog

1. Installation

    - GIT auf Ubuntu/Debian installieren
    - GIT unter Windows installieren

2. Integrations (git)

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

3. Other GUIs

    - git extensions gui
    - gui uebersicht

4. testing submodules (v2)

Geschichte / Grundlagen

GIT Pdf

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

Daily work (best practices)

Best practices für daily work mit feature-branches

Szenario 1: Arbeiten an einem Feature

- Wann immer ich etwas umsetzte, erstelle ich zunächste ein feature
- Arbeitszeit an Feature möglichst klein halten (wenn feature zu gross, dann in kleinere mini-features aufbrechen)

Schritt 1: Am Tag 0

1. Im master (ich muss mich im master befinden) den aktuellen Stand herstellen (tortoise -> pull -> master)
2. In einen neuen Branch reinswitchen (tortoisegit -> switch/checkout -> create branch) Empfehlung Prefix feature/ z.B. feature/anpassung-instrument oder auch ticket-id
3. An dem Projekt arbeiten (d.h. add + commit von neuen und geänderten files), gerne viele commits pro tag
4. einmal täglich Stand auf server pushen (für Krankheitsfall) - push des feature-branches

Schritt 2: Danach täglich (wenn ich länger an dem Feature arbeite)

1. morgen: aktuellen stand holen: tortoisegit -> pull vom master [remote branch : master) (ich bin nachwievor im Feature)
2. an dem projekt arbeiten (s. Schritt 1)
3. abends wieder stand auf server push (für Krankheitsfall) push des feature - branches

Schritt 3: Juppie Yeah, ich bin fertig mit feature

1. Nochmal zur Sicherheit Änderungen vom master pullen (ich bin nachwievor im feature) (d.h. im feature -> tortoisegit -> pull (remote branch: master) Evtl. hier Konflikte auflösen
2. Pushen des feature-branches auf den Server
3. Merge request erstellen (Achtung: Source ist der feature - branch, target is der master branch --> d.h. Änderung sollen in den master übernommen werden)

Schritt 4a: mergen (kein Konflikt vorhanden) in gitlab

- Bevorzugt online mergen
- Im merge request ist ersichtlich, dass ich mergen kann (grüner Mergebutton erscheint)

1. Gitlab: Merge button klicken (es wird gemerged)
2. Aufräumen (Tortoisegit)
    1. In master wechseln (tortoisegit -> switchto -> master)
    2. Pull im master von master (remote branch bleibt master)
    3. tortoisegit -> show log -> lokalen branch löschen

Oder: Schritt 4b: mergen (konflikt vorhanden) in gitlab

- Bevorzugt online mergen
- Im merge request ist ersichtlich, dass ich NICHT mergen kann (rotes Schild, kein grüner Button vorhanden)

1. tortoisegit im feature-branch -> tortoisegit -> pull -> von master !! (Achtung: remote branch in master) ändern im Select
2. tortoisegit: Es wird ein Konflikt angezeigt und der Resolve button (ganz links). Diesen drücken
3. tortoisegit: Kleines Fenster öffnet sich mit Liste der Files mit Konflikt in rot
4. tortoisegit: File anklicken(doppelt)
5. tortoisgit: mergetool öffnet sich
6. tortoisegit mergetool: Konflikte lösen, danach grünen Haken drücken und Fenster schliesen
7. tortoisegit: kleines Fenster schliessen
8. tortoisegit: commit -> feature/xyz (Achtung kommentare rausnehmen und commit klicken)
9. tortoisegit: push -> feature/xyz - branch
10. Online: merge request neu laden (F5) oder einfach nochmal im Menü links merge request anklicken
11. Online: im merge request - mergen (sollte jetzt auf grün stehen)

12. Aufräumen (Tortoisegit)
    1. In master wechseln (tortoisegit -> switchto -> master)
    2. Pull im master von master (remote branch bleibt master)
    3. tortoisegit -> show log -> lokalen branch löschen

Installation / Nutzung (tortoisegit)

Installation tortoisegit

Reihenfolge

Schritt 1: Runterladen

```
1. Notepad++
https://github.com/notepad-plus-plus/notepad-plus-
plus/releases/download/v8.6.9/npp.8.6.9.Installer.x64.exe
2 git for windows
https://github.com/git-for-windows/git/releases/download/v2.45.2.windows.1/Git-2.45.2-64-
bit.exe
3. Lanugage Pack tortoisegit (deutsch)
https://download.tortoisegit.org/tgit/2.16.0.0/TortoiseGit-LanguagePack-2.16.0.0-64bit-de.msi
4. Tortoisegit
https://download.tortoisegit.org/tgit/2.16.0.0/TortoiseGit-2.16.0.0-64bit.msi
```

Schritt 2: Installation

- Die Pakete 1 bis 4 einfach durchklicken
- Ausser: Tortoisegit, bei Editor, Editor angeben

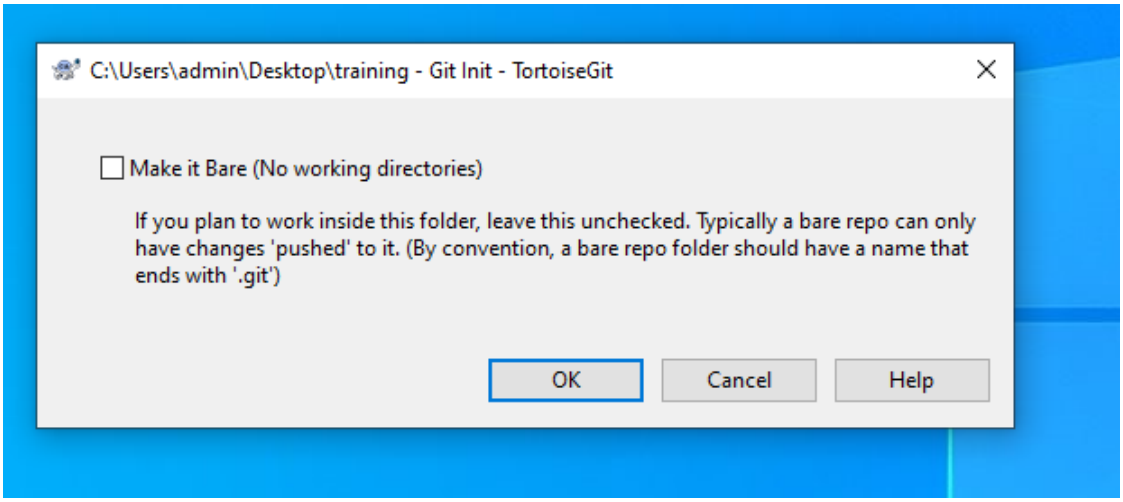
Neues Repo einrichten

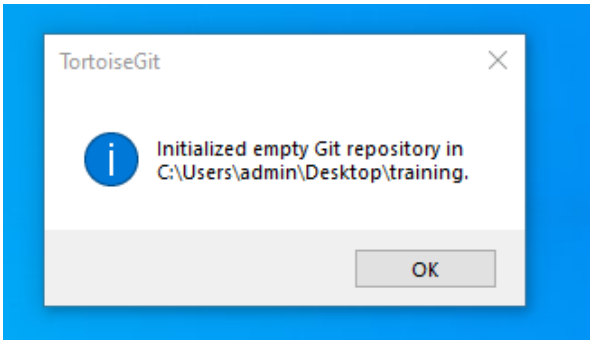
Step 1: Create folder for projects or use given folder

- training (auf dem Desktop)

Step 2:

Step 3: Keinen Haken setzen --> (nur für Server)





tortoisemerge für git bash einrichten

Find out if mergetool TortoiseMerge is available

```
## tortoisegit is already installed
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 tortoisemerge
git config --global diff.tool tortoisemerge
git config --global mergetool.keepBackup false
git config --list
```

How to use it

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

Fehlende Icons anzeigen - FIX

Walkthrough

```
Schritt 1:
Registrierungseditor öffnen

Schritt 2:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers

Alle Einträge von Tortoise durch umbenennen (mehrere Leerzeichen davorstellen)

Schritt 3:
Task-Manager öffnen und Windows - Explorer neu starten
CTRL + SHIFT + ESC


Hinweise
https://stackoverflow.com/questions/25156238/tortoisegit-not-showing-icon-overlays
```

Commands git (with tipps & tricks)

git alias

```
git config --global alias.cc 'commit'
```

Special Alias (multiple commands) -

```
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

* nonsene commit-message becasue of missing text-expertise"
## enter on last line
```

Change last commit-mesage (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 resultion 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 FORMAT)
- 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
git pull --rebase

## 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 practive

```
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
```

conflict you have in merge-request (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"

## Uups, 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 specfic 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 (not free)
- 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
```

Tipps & Tricks (Tags)

Delete Tags being deleted online also locally

```
git fetch --prune --prune-tags
```

Tipps & Tricks (.gitignore)

Exercise: fix already committed files for .gitignore

- Want to ignore them

Steps Overview

1. Delete files/Folder from Repo
2. Add files/folder to .gitignore
3. (Push to server)
4. (All people in the team, pull new .gitignore)

Exercise

Step 1 Ordner mit objekt-file in repo packen (Sorry ! Versehentlich)

```
1. wie legen einen debug ordner an
2. und in dem debug-ordner packen wie 3 objekt.

datei1.obj
datei2.obj
datei3.obj

3. add + commit -> tortoisegit -> commit
```

Step 2 .gitignore

```
-> dateien rauslöschen aus repo durch anklicken ordner
1. Datei aus dem Ordner auf dem Repo löschen (delete)
auf den ordner debug rechte Maustaste ( TortoiseGIT /  Delete)

2. commit -> master
uncheck all
und nur die delete einträge anhaken

3. commit - button

--> ab sofort nicht mehr erfassen von git

4. Ordner ignorieren
rechte Maustaste ordner -> tortoise git -> add to ignorelist
o only folder
```

```
o put .gitignore  in root
Ok

5. Testen: git commit -> master
super -> nicht mehr da

.gitignore adden und committen

6. Testen: Fügen mal eine neue test.exe

7. Wird sie ignoriert -> git commit -> master
anschauen
```

Editoren / Editoren konfigurieren

Editoren RapiOS bzw. Raspian

```
#### Visual Studio Code
https://code.visualstudio.com/docs/setup/raspberry-pi
GitUI
https://pi-apps.io
List of editors
https://pi-apps.io/wiki/getting-started/apps-list/
```

Notepad++

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 - KOCHREZEPT

```
## 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

## modfizieren 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 checkout master
git pull --prune master
git branch -D feature/5021
```

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 bitbucken - 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
```

Mergetool

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
```

Using tortoisegit as a mergetool to solve conflicts

Step 1: Install tortoisemerge

- https://tortoisegit.org/download/

Step 2: Close git bash if open

Step 3: Does git find merge

```
git mergetool --tool-help
```

Step 4: Setup tortoisemerge as mergetool

```
git config --global merge.tool tortoisemerge
git config --global mergetool.keepBackup false
```

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

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 fü+r 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 kompiziert (Lernkurve größe 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)
- Benamung: 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

substrees

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 practive

```
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

submodules in tortoisegit

submodules in tortoisegit

Exercise 2.1 submodule einbinden

```
url: https://github.com/jmetzger/training-tortoisegit
branch: main

1. submodule add
https://github.com/jmetzger/training-tortoisegit
main
Path: training-tortoisegit

OK

2. committed
```

Exercise 2.2 Bibliothek aktualisieren

```
1. Jochen: Online geändert
2. Änderung lokal übernehmen
2.1 für den Ordner pull
```

2.2. Elternordner (Projekt training: commit)

Authentication

Work with different credentials

Ref:

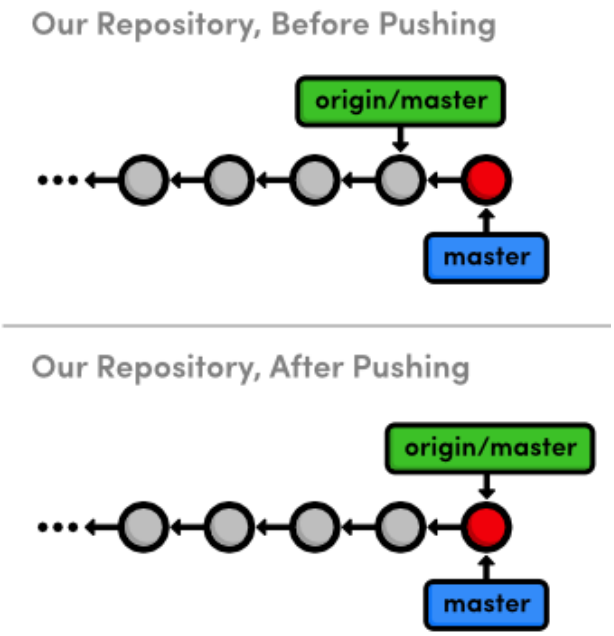
https://de.linkedin.com/pulse/mehrere-gitlabgithub-accounts-bzw-ssh-keys-zum-host-mit-mindermann

Tortoise - Documentation

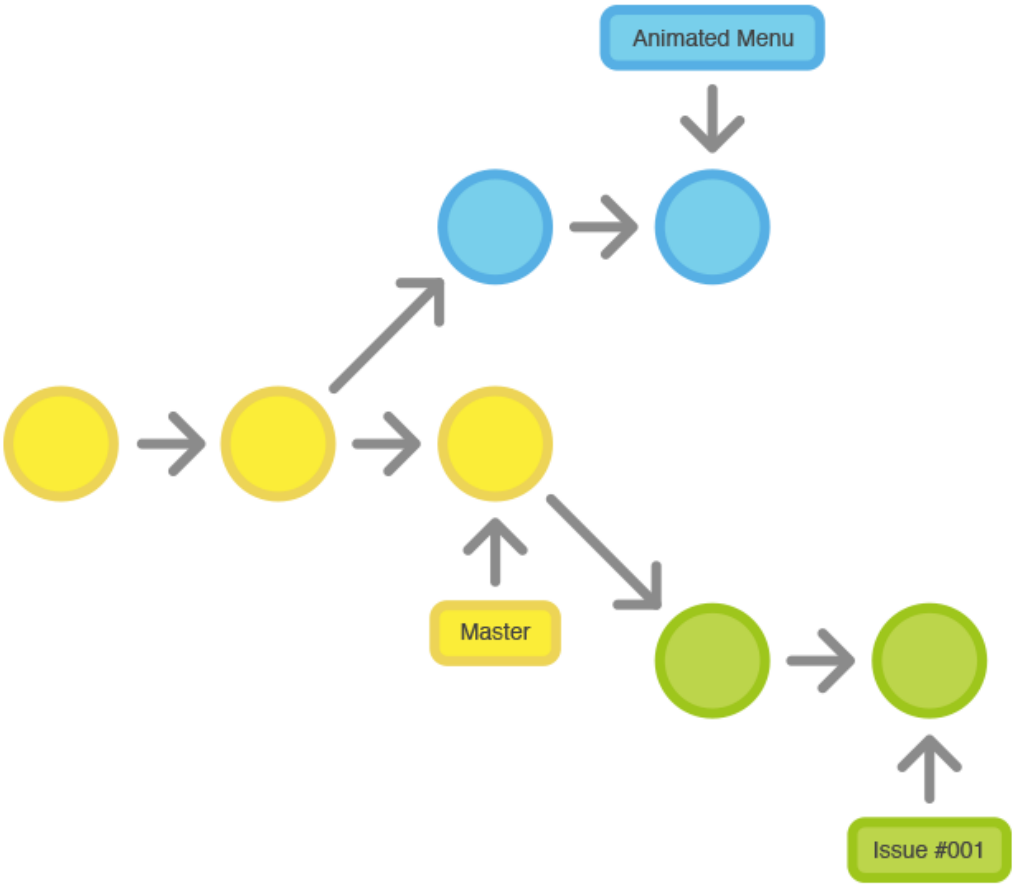
Remote in Teams arbeiten

Workflows

Centralized Workflows



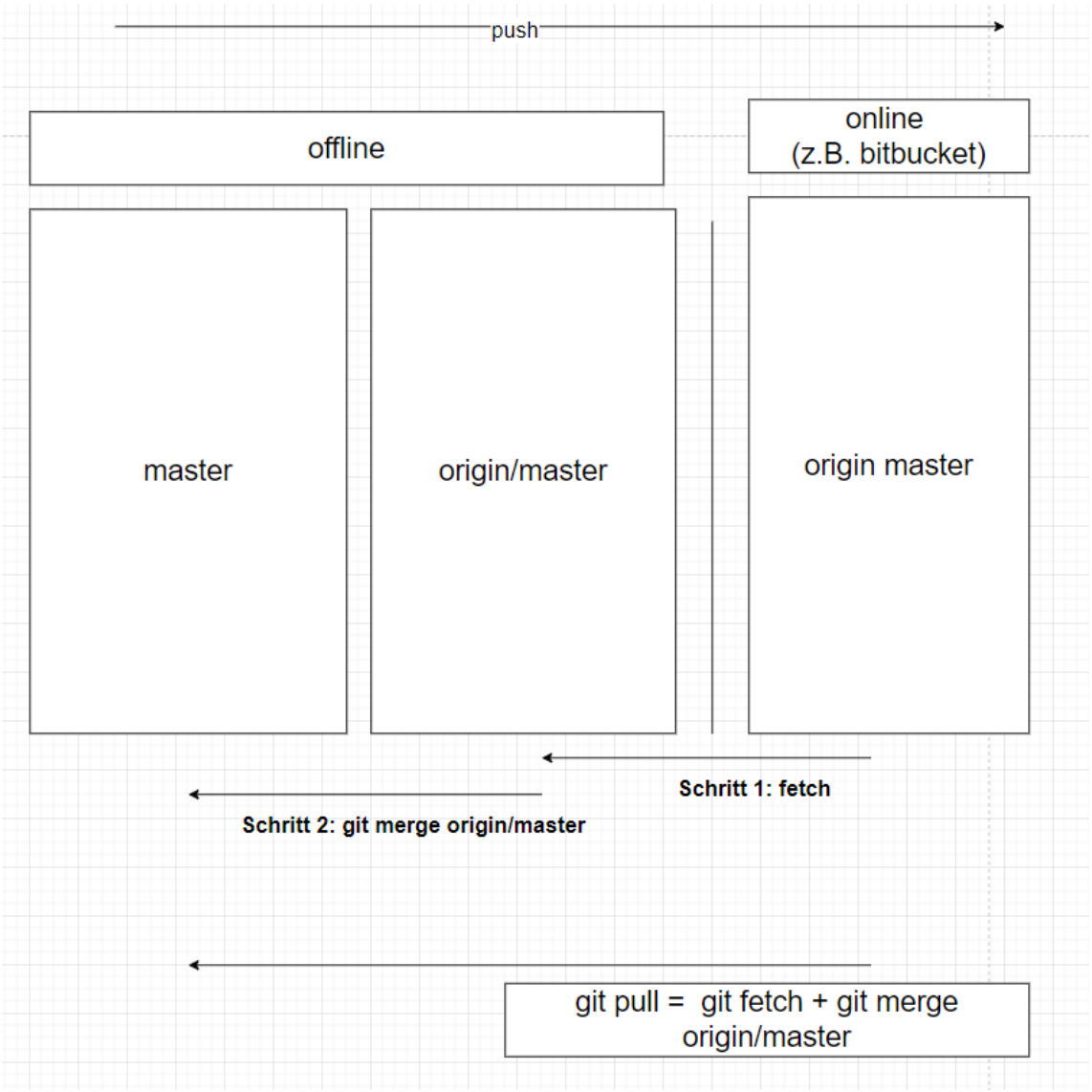
feature workflow



gitflow workflow

image

Branches and Onlinebranches



Fix conflict in pull-request - gitlab

Walkthrough

```
## create feature-branch and worked on it
git checkout -b feature/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, otherwice, 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
```

Documentation - .gitignore

List for different programming lanuages

- https://github.com/github/gitignore

Documentation - permissions

Permissions

- https://docs.gitlab.com/user/permissions/

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 / Continous 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/

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

Integrations (git)

Other GUIs

git extensions gui

Installation

- http://gitextensions.github.io/

gui uebersicht

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

testing submodules (v2)