GIT-Training (mit Intellij)

Agenda

1. Geschichte / Grundlagen
    - GIT Pdf
2. Intellij
    - Neues Projekt erstellen - ohne git
    - Bestehendes Projekt unter GIT Versionsverwaltung stellen
    - Git Untermenü in obererer Menüleiste einrichten
    - Disable ESC when using vi as editor
3. Intellij - Tools
    - Easygitlab - Erweiterung für merge request und pipeline management
4. Intellij - Tipps & Tricks
    - Nicht automatisch mergen - d.h. kein merge-commit
5. Intellij - Übungen
    - Übungen
6. JIRA/Gitlab - Integration
    - Jira-Integration
7. Commands (with tipps & tricks)
    - 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 push/pull
    - git reset
    - git rm (Dateien löschen aus git)
8. Branches / Branching /workflows
    - GIT Workflows
    - Branch Overview with origin image
9. Advanced Commands
    - git reflog
    - git reset - Back in Time
10. 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 specfic commit message
- Alle Dateien, die sich geändert haben anzeigen z.B. heute

11. Tipps & Tricks (editor)

    - Notepad als Editor verwenden- Windows
    - TextEdit als Editor unter mac verwenden

12. Tipps & Tricks (Aufräumen)

    - Tracking Branches (shadow branches) nach Integration Online löschen

13. Exercises

    - merge feature/4712 - conflict
    - merge request with bitbucket

14. Snippets

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

15. Extras

    - Best practices
    - Using a mergetool to solve conflicts

16. Help

    - Help from commandline

17. submodules

    - submodules

18. Authentication

    - Work with different credentials

19. Shells

    - color for zsh-shell under osx
    - branch mit anzeigen in zsh-shell und osx

20. 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
- .gitignore
- learn branching

21. Datenbank - Versionierung

    - Methode 1
    - Methode 2

22. Extra / Gitlab CI/CD

    - Kann ich einen stage manuell starten

Backlog

1. Installation
    - GIT auf Ubuntu/Debian installieren
    - GIT unter Windows installieren

Geschichte / Grundlagen

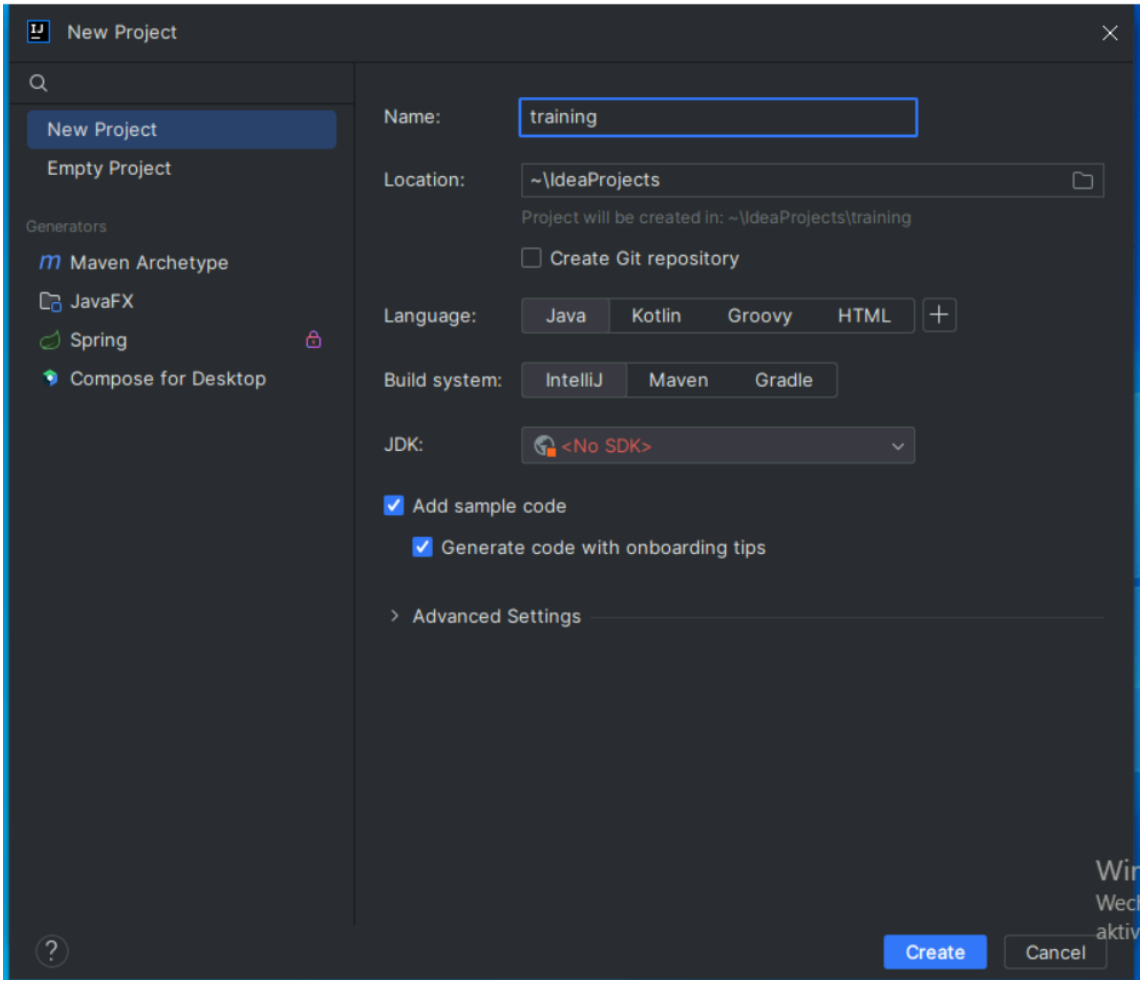
GIT Pdf

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

Intellij

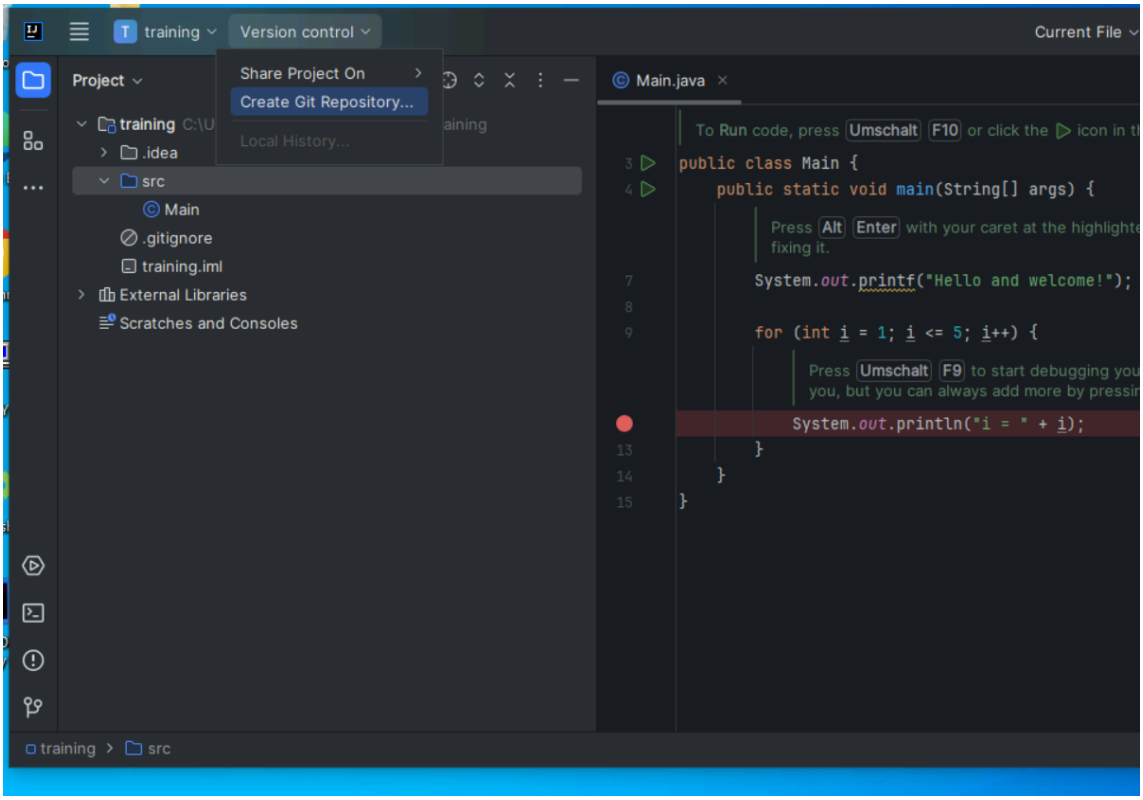
Neues Projekt erstellen - ohne git

1. Projektname "training"
2. Alles andere lassen
3. SDK -> runterladen und auswählen (JDK)
4. Create

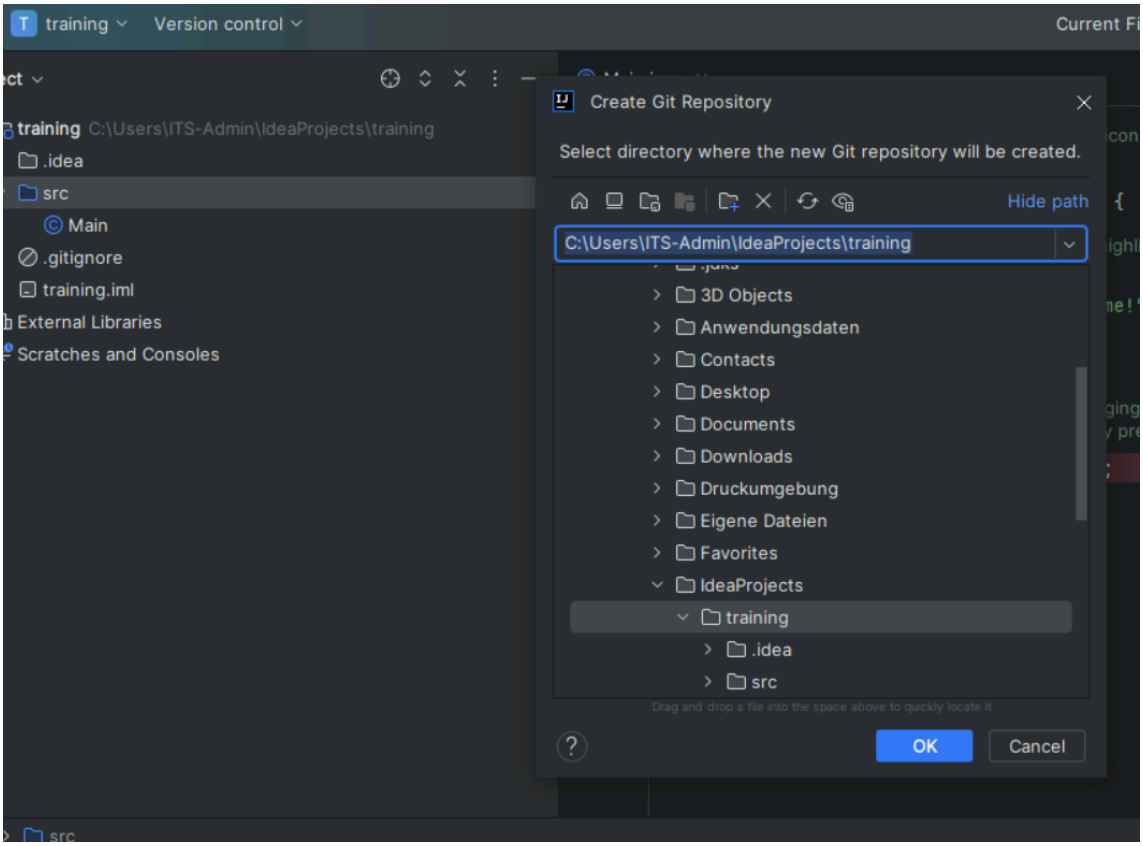


Bestehendes Projekt unter GIT Versionsverwaltung stellen

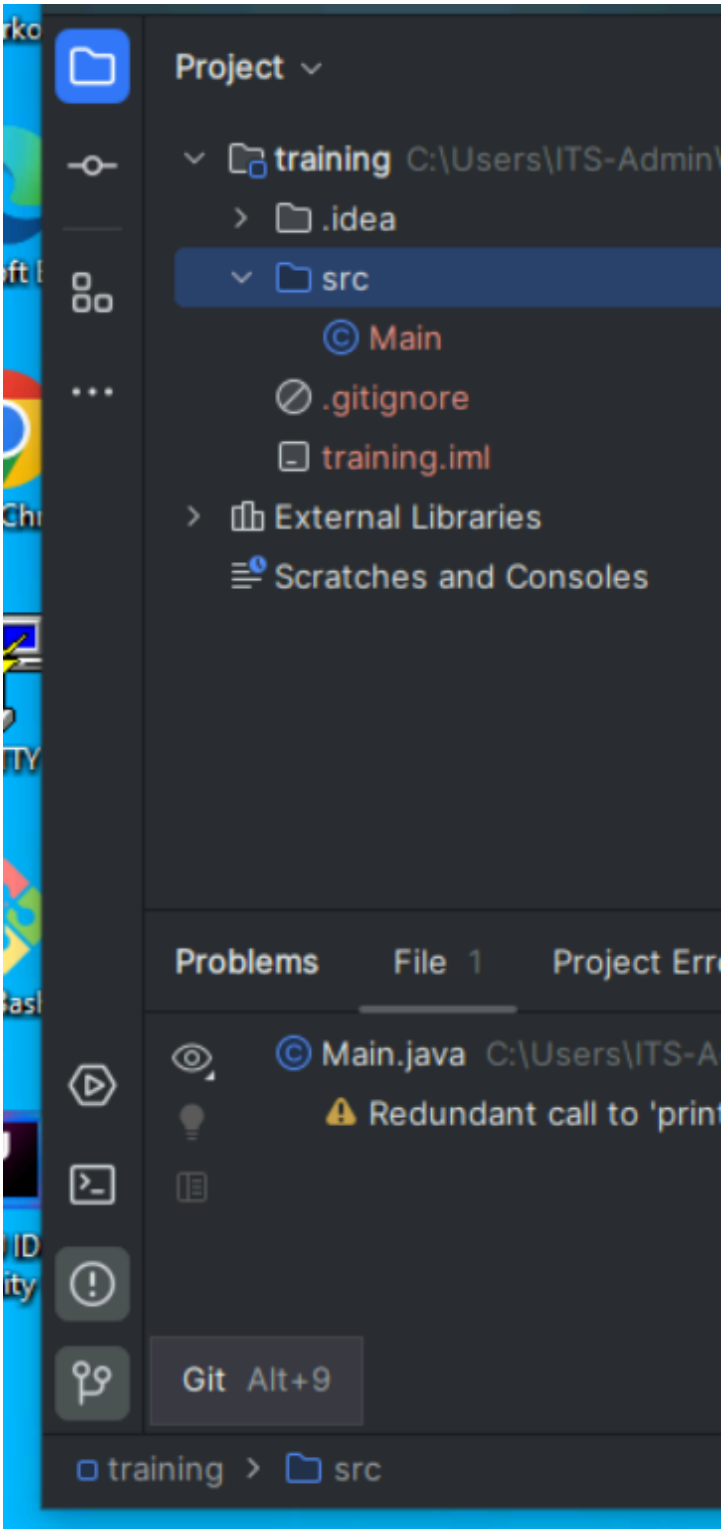
Schritt 1: Oben links neben Projekt -> Version Control (Seletor anklicken)



Schritt 2: Namen des Projektes übernehmen

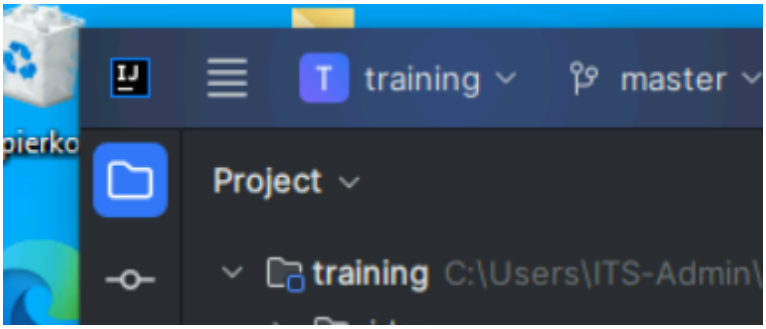


Schritt 3: Versionsverwaltung steht jetzt unten links zur Verfügung (Seitenmenü)



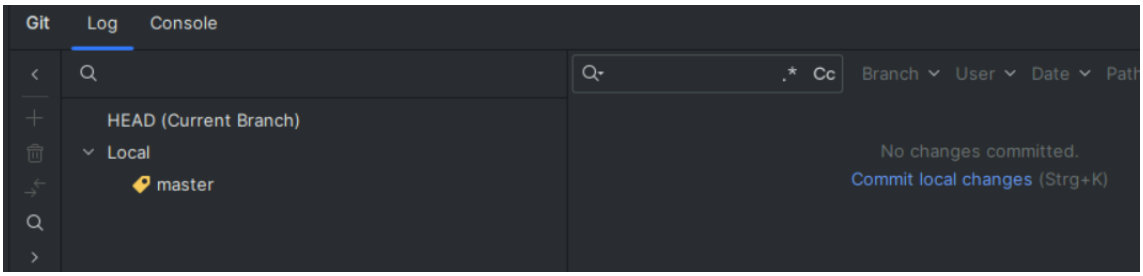
Schritt 4: Anklicken ;o)

Schritt 5: Falls oben im linken Menü -o- noch nicht sichtbar



Dann:

Entweder STRG + k - Tastaturkombinationen verwenden oder auf commit-local changes klicken

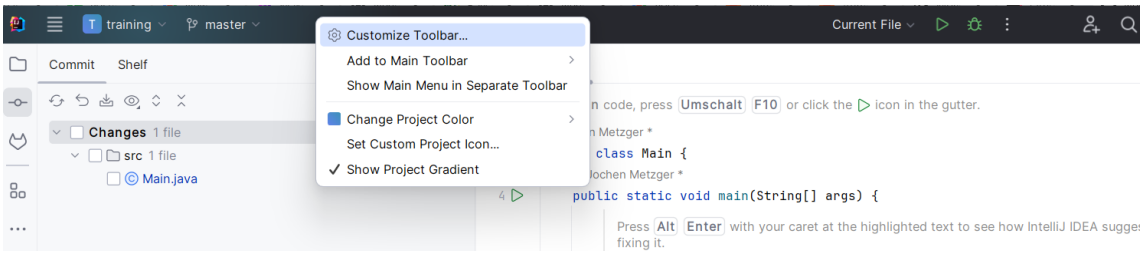


Wichtig: Bestimmte Files sollten ignoriert werden.

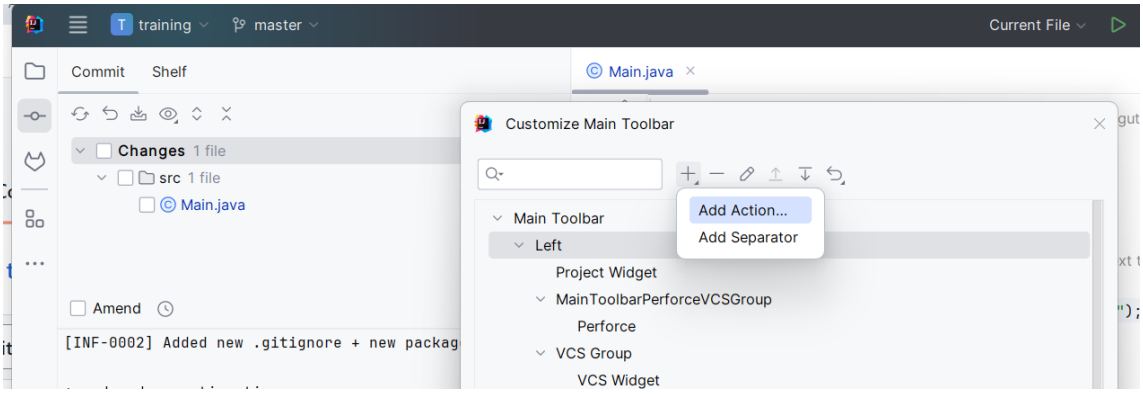
- https://intellij-support.jetbrains.com/hc/en-us/articles/206544839-How-to-manage-projects-under-Version-Control-Systems
- Das passiert in neueren Version bereits automatisch. (seit 2019.1)

Git Untermenü in obererer Menüleiste einrichten

Rechte Maustaste

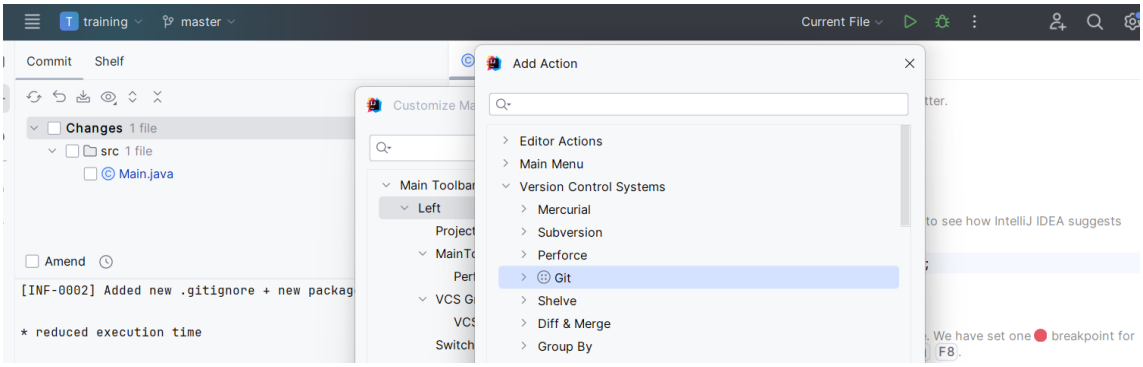


Neue Action hinzufügen

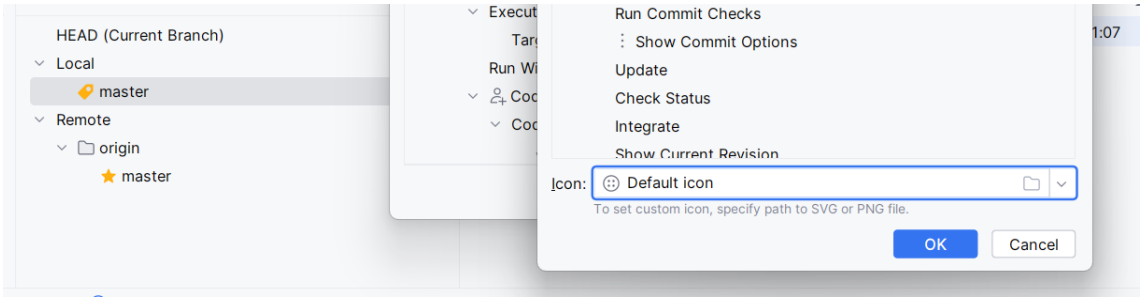


Git anhaken

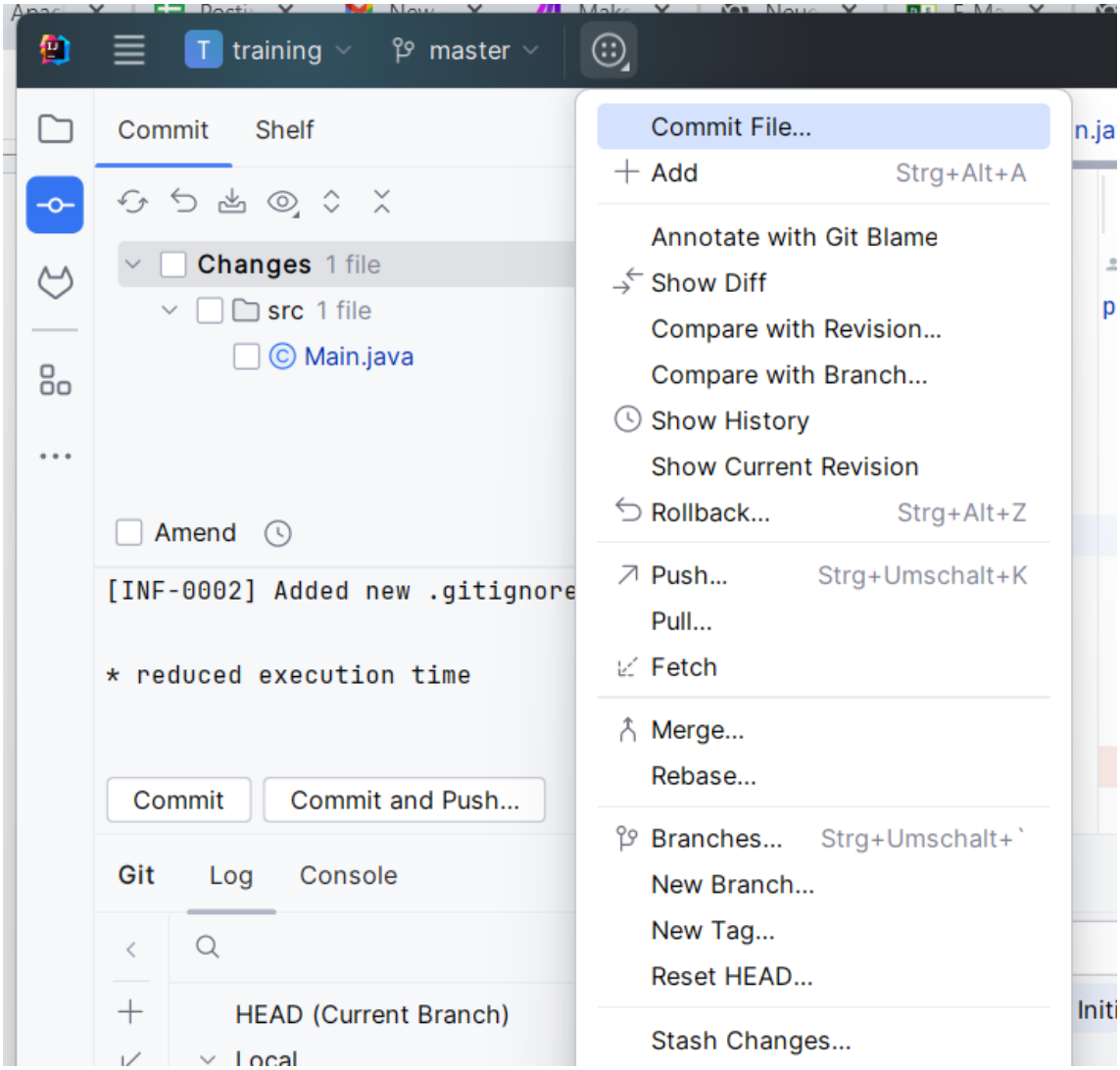
- 3. Punkt -> Version Control System -> git



- Dann unten rechts OK
- -> Danach im nächsten Fenster nochmal o.k.



Git Symbol erscheint



Disable ESC when using vi as editor

- https://intellij-support.jetbrains.com/hc/en-us/community/posts/360003508579-How-to-stop-Escape-from-Leaving-Terminal

Intellij - Tools

Easygitlab - Erweiterung für merge request und pipeline management

- https://plugins.jetbrains.com/plugin/19611-easy-gitlab

Intellij - Tipps & Tricks

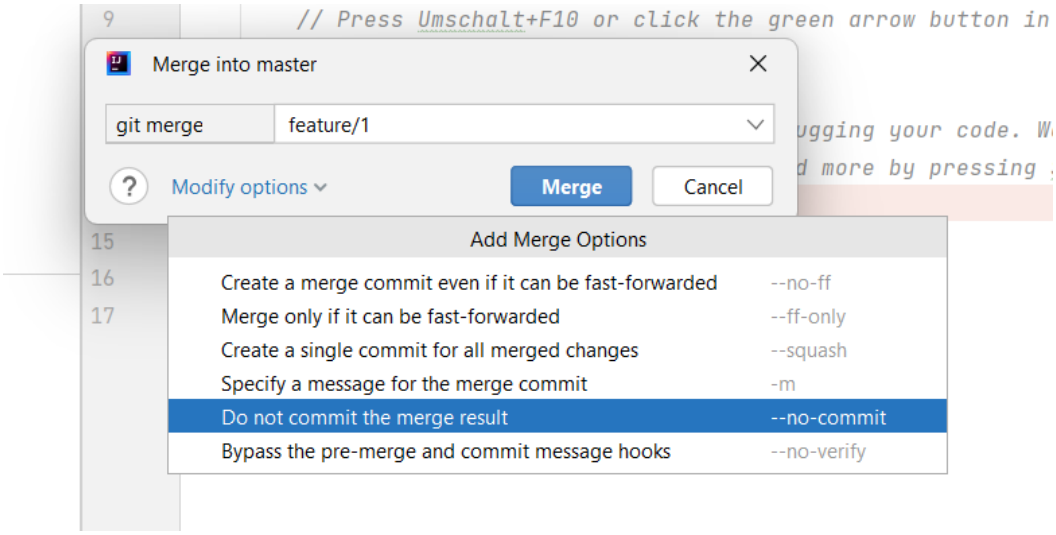
Nicht automatisch mergen - d.h. kein merge-commit

Why ?

- If you want to review the files before commit - message is done

Where ?

- When you want to merge -> in top menu -> GIT -> Merge...
- Then ->



On the commandline

```
git merge --no-commit <your-branch-to-be-merged>
```

Intellij - Übungen

Übungen

```
### Übung 11

1. Trainer legt Gruppenprojekt an und pushed code
2. Gruppenprojekt clonen
3. Zugriff über gitlab testen:
https://gitlab.com/training.tn1/bbgruppe1/
4. feature-branch erstellen

feature/jochen1

5.  eine neue datei anlegen + adden +committen

6. feature  - branch pushen

7. merge-request


--- nacheinander
8. mergen

--- aufräumen
9. wechseln in den master
10. master pull (löscht er den schattenbranch)
```

11. feature - branch

Übung 10

1. lokal: neuen Branch feature/4840
2. Ändern README.md Zeile 1 + add + commit
3. Ändern datei2.txt + add + commit

4. Online in master: neuen ánhänge readme
committen

5. Online in master: Zeile 1 readme
committen

6. Änderungen von Online mit pull --rebase abholen

---

7. dann feature pushen

8. merge request

9. Aufräumen
a. in master wechseln
b. git pull master
c. feature-branch löschen

Übung 9c

1. feature/4836 pushen
2. merge requests
3. mergen

3.5 Anschauen im graphen online

4. aufräumen
a. in den master wechseln
b. pull --prune
c. feature löschen

Übung 9b

1. lokal: neuen Branch feature/4836
2. Ändern datei1.txt + add + commit
3. Ändern datei2.txt + add + commit

4. Online in master: neuen ánhänge readme
committen

```
5. Online in master: neuen ánhänge readme
committen

6. Änderungen von Online mit pull --rebase abholen

---

7. dann feature pushen

8. merge request

9. Aufräumen
a. in master wechseln
b. git pull master
c. feature-branch löschen


### Übung 9a

1. lokal: neuen Branch feature/4835
2. Neue datei1.txt + add + commit
3. neue datei2.txt + add + commit

4. Online in master: neuen ánhänge readme
committen

5. Online in master: neuen ánhänge readme
committen

6. Änderungen von Online mit pull (merge) abholen

7. dann feature pushen

8. merge request

9. Aufräumen
a. in master wechseln
b. git pull master
c. feature-branch löschen


### Übung 8: neues feature mit merge-request (mit Konflikt)

1. Online in master -> README.md Zeile 1 ändern
--

2. Lokal: Neuen branchen feature/4832
3. Ändern Datei README.md Zeile 1 ändern + add + commit
4. git push -u origin feature/4832

5. Online: Merge Request  erstellen
```

```
-> Conflict

5.5. Konflikt lösen
a. Die Änderung lokal abholen (vom master)
b. Conflict lösen
c. Mergen lokal
d. noch pushen


5.6. und mergen


6. Aufräumen
6.1. master wechseln udn auf 'n Stand bringe
6.2. pull --prune // schattenbranch löschen (Remote Tracking Branch)
6.3.  lokalen feature-Branch löschen




### Übung 7: neues feature mit merge-request (ohne Konflikt)

1. Lokal: Neuen branchen feature/4831
2. Lokal neue Datei f1.txt erstellen + add + commit
3. Lokal neue Datei f2.txt erstellen + add + commit
4. git push -u origin feature/4831

5. Online: Merge Request  erstellen
5.5. und mergen


6. Aufräumen
6.1. master wechseln
6.2. pull --prune // schattenbranch löschen (Remote Tracking Branch)
6.3.  lokalen feature-Branch löschen
c. master auf Stand bringe



### Übung 6:(Online und lokale Änderung)

1. Online: README ändern -> Zeile 1
2. Lokal Datei README -> Zeile 1
+ add + commit
3. Push
4. Konflikte  lösen
5. Nochmal pushen


### Übung 5:(Online und lokale Änderung)
```

1. Online: README ändern
2. Lokal neue Datei AGENDA.md mit einer Zeile
+ add + commit
3. Push

Übung 4: (Onloneänderung ohne Konflikt)

    1. Online: Ändern der README
    2. Lokal: Änderung abholen

Übung 3: Übung reset

1. Auf alten Stand zurück in intellij über log
2. auf kommandozeile gehen und mit reflog letztes commit
vor reset ausfindig machen
3. git reset --hard <zu-id-aus-2>
4. Überprüfung des Logs in Intellij

Übung 2: Konflikt mit mergetool

1. You are in master-branch
2. Checkout new branch feature/4722
3. Change line1 in README.md
4. git add -A; git commit -am "done"
5. Change to master
6. Change line1 in README.md

7. git add -A; git commit -am "change line1 in README in master"

8. git merge feature/4722

Übung 1: Konflikt

1. You are in master-branch
2. Checkout new branch feature/4721
3. Change line1 in README.md
4. git add -A; git commit -am "feature-4721 done"
5. Change to master
6. Change line1 in README.md

7. git add -A; git commit -am "change line1 in todo.txt in master"

```
7.5. Tiefsinnige Betrachtung -> Log


8. git merge feature/4721
```

JIRA/Gitlab - Integration

Jira-Integration

Reference

- https://docs.gitlab.com/ee/integration/jira/development_panel.html
- Jira -> DataCenter
  - https://docs.gitlab.com/ee/integration/jira/development_panel.html#use-the-integration
- https://about.gitlab.com/blog/2021/05/24/gitlab-and-jira-integration-the-final-steps/

Commands (with tipps & tricks)

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
```

Die verschiedenen Ebenen

```
## user global gesetzt
git config --global user.name "Jochen Blaumann Metzger"
##nur im Repo gesetzt
git config user.name "J. Blaumann"
## Zeig mal was auf Systemebene gesetzt ist (oberste Ebene)
git config --list --system
## Zeige mal, was auf User-Ebene gesetzt ist (2. Ebene)
git config --list --global
## Zeig mal was auf Repo-Ebene gesetzt ist
git config --list --local
##ä Und was ist das ergebnis ? -> per Vererbung, was nimmst du am Ende
git config --list
```

```
## Weclhe email wird jetzt im Repo verwendet
git config user.email
## Welcher Name wird jetzt im Repo verwendet ?
git config user.name
```

git show

Show information about an object e.g. commit

```
## Show commit info and what has changed
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
```

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

```
## 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 push/pull

Neuen Branch pushen

```
git push -u origin neuer-branch
```

Unseren lokalen master auf dem Stand mit online master halten

```
git pull --rebase origin master
```

Wir arbeiten an einem branch und wollen diesen täglich mit dem master aktualiseren

```
## Initial beim ersten mal damit arbeiten
git checkout master
git checkout -b feature/neuer-branch

## danach täglich im feature/neuer-branch
git pull --rebase origin master
```

git reset

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
```

Example (Arbeitsumgebung auf den Stand des letzten Commits setzten)

```
## linux befehl -legt leere datei an.
touch myfile
git add myfile
## ich will dieses file nicht !!!!!
git reset --hard HEAD
```

git rm (Dateien löschen aus git)

Datei komplett löschen (Workspace, Index und Repo)

```
git rm  dateiname
```

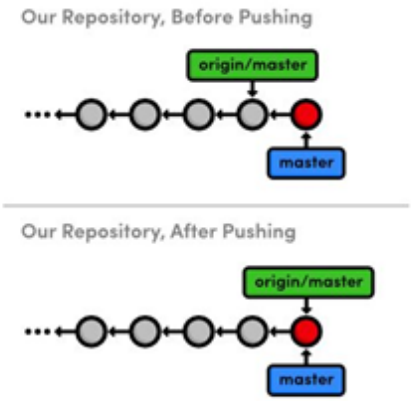
Datei nur aus Repo und Index löschen

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

Branches / Branching /workflows

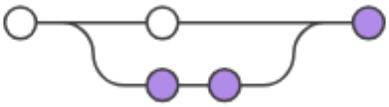
GIT Workflows

Centralized Workflow



Pushing master to the central repository

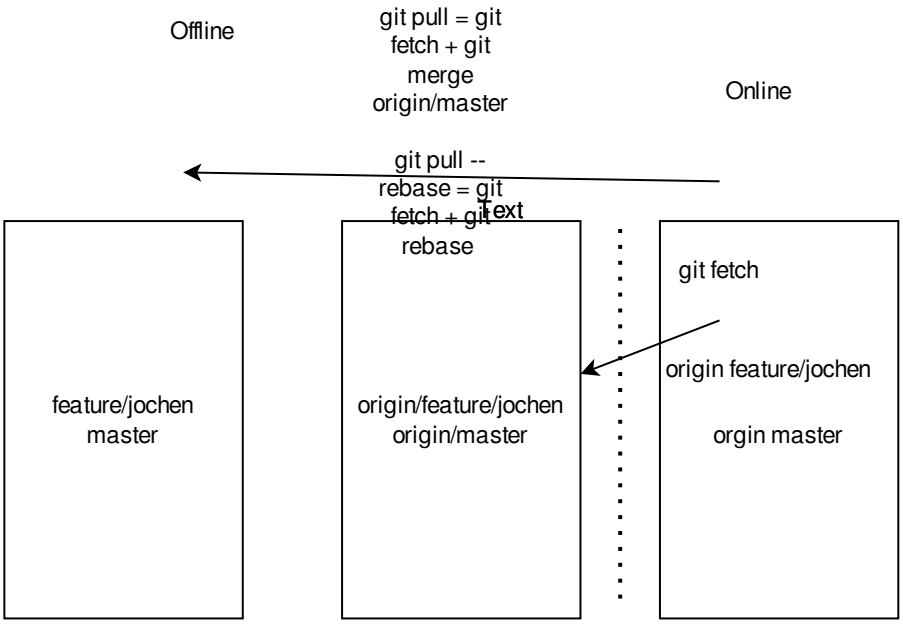
Feature Workflow



gitflow workflow

Branch Overview with origin image

master vs. origin/master vs origin master



Advanced 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
```

Example (Arbeitsumgebung auf den Stand des letzten Commits setzten)

```
## linux befehl -legt leere datei an.
touch myfile
git add myfile
## ich will dieses file nicht !!!!!
git reset --hard HEAD
```

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

## 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
```

Always rebase on pull - setting

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

Arbeit mit submodules

Walkthrough

```
##
## uses the same branch as main repo
git submodule add https://github.com/jmetzger/training-git-pycharm

## tracking is done here:
## cat .gitmodules
```

clone repo with submodules (done afaik automatically by pycharm)

```
git clone --recurse-submodules --remote-submodules <repo-url>
```

Updating commands for updating subfolder

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

Task: Set submodule to a specific tag

```
cd submodule_directory
git checkout v1.0
cd ..
git add submodule_directory
git commit -m "moved submodule to v1.0"
git push
```

pycharm notes

- There is no action for "git submodule add" in IDE
- Once you do it from CLI, you can add it to PyCharm's Version Control here: `Version Control > Directory Mappings`

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, 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
```

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 (editor)

Notepad als Editor verwenden- Windows

```
git config --global core.editor notepad
```

TextEdit als Editor unter mac verwenden

```
git config --global core.editor "open -W -n"
```

Tipps & Tricks (Aufräumen)

Tracking Branches (shadow branches) nach Integration Online löschen

```
git checkout master
git pull --rebase origin master

## Davor: Online branch (neuer feature-branch löschen)
## Löscht Schattenbranch
git fetch --prune

## Und lokaler branch löschen
git branch -d feature/mein-feature-branch
```

Exercises

merge feature/4712 - conflict

Exercise

```
1. You are in master-branch
2. Checkout new branch feature/4712
3. Change line1 in README.md
4. git add -A; git commit -am "feature-4712 done"
5. Change to master
6. Change line1 in README.md
7. git add -A; git commit -am "change line1 in todo.txt in master"
8. git merge feature/4712
```

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

Cleanup locally

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

```
 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
```

Help

Help from commandline

On Windows

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

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

submodules

submodules

Walkthrough

```
##
## uses the same branch as main repo
git submodule add https://github.com/jmetzger/training-git-pycharm

## tracking is done here:
## cat .gitmodules
```

clone repo with submodules (done afaik automatically by pycharm)

```
git clone --recurse-submodules --remote-submodules <repo-url>
```

Updating commands for updating subfolder

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

Task: Set submodule to a specific tag

```
cd submodule_directory
git checkout v1.0
cd ..
git add submodule_directory
git commit -m "moved submodule to v1.0"
git push
```

pycharm notes

- There is no action for "git submodule add" in IDE
- Once you do it from CLI, you can add it to PyCharm's Version Control here: `Version Control > Directory Mappings`

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

Shells

color for zsh-shell under osx

- https://gist.github.com/chrisnolet/d3582cd63eb3d7b4fcb4d5975fd91d04

branch mit anzeigen in zsh-shell und osx

- https://github.com/romkatv/powerlevel10k

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/

.gitignore

- https://git-scm.com/docs/gitignore

learn branching

- https://learngitbranching.js.org/?locale=de_DE

Datenbank - Versionierung

Methode 1

- https://github.com/sergiosbx/pyway

Methode 2

- https://flywaydb.org/

Extra / Gitlab CI/CD

Kann ich einen stage manuell starten

So geht's ;o)

```yaml
stages:          # List of stages for jobs, and their order of execution
  - build
  - test

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
```

```
    - echo "Compile complete."

unit-test-job:   # This job runs in the test stage.
  stage: test    # It only starts when the job in the build stage completes
successfully.
  script:
    - echo "Running unit tests... This will take about 60 seconds."
    - sleep 60
    - echo "Code coverage is 90%"
  when: manual
```

Beispiel: Deploy läuft erst, wenn Testing erfolgreich durchgeführt wurde

- d.h. händisch gestartet und erfolgreich ausgeführt wurde.
- Keyword ist hier: allow_failure: false

```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test
  - deploy

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."

unit-test-job:   # This job runs in the test stage.
  stage: test    # It only starts when the job in the build stage completes
successfully.
  script:
    - echo "Running unit tests... This will take about 60 seconds."
    - sleep 10
    - echo "Code coverage is 90%"
  allow_failure: false
  when: manual

deploy-job:   # This job runs in the test stage.
  stage: deploy    # It only starts when the job in the build stage completes
successfully.
  script:
    - echo "Now deploying "
    - echo "Deployment done "
```

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