

# Eclipse with (E)git Training

## Agenda

### 1. Technical Background

- [Projects with git](#)
- [How does it work?](#)
- [The flow of git](#)
- [Git and its objects](#)
- [5-goldene-Regeln - Zerstörung](#)

### 2. Commands - Commandline (with tips & 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 remote -v](#)

### 3. Eclipse/STS(Spring Tool Suite)

- [Increasing Icon Size](#)
- [Installation of Eclipse](#)
- [Installation of STS \(Spring Tool Suite\)](#)
- [Prepare Perspective for good user experience](#)
- [Create/Initialize Repository/Project](#)
- [Eclipse Icons](#)
- [Git-Commands in eGit](#)
- [Setup Identity](#)
- [Staging Area](#)
- [Log](#)
- [Commit](#)
- [Add](#)
- [Edit markdown in eclipse with plugin](#)

### 4. Advanced Commands

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

### 5. Features

- [Partial Clone](#)
- [Hooks](#)

### 6. GIT-Server

- [GIT Server](#)

### 7. Golden

- [5-goldene-Regeln](#)

#### 8. Tipps & Tricks (git / git bash)

- [Konfigurationseinstellung löschen](#)
- [Anderen Editor verwenden](#)
- [Platz sparen mit dem shallow clone + neu pushen](#)
- [Branch aus bestehendem Branch neu erstellen \(alles auf Anfang!\)](#)
- [Arbeiten mit git replace](#)
- [ssl Verifizierung ausschalten \(quick & dirty\)](#)

#### 9. GIT Mergetools

- [git mergetools](#)

#### 10. GIT-Guis

- <https://git-scm.com/download/gui/windows>
- <https://github.com/DmitryZheltnin/git-extensions-intellij>
- <https://github.com/gitextensions/gitextensions/>
- <https://git-fork.com/>

#### 11. Exercises

- [Create simple conflict on commandline](#)
- [Conflict merge-request](#)

#### 12. Documentation

- [GIT pdf](#)
- [http://wiki.eclipse.org/EGit/User\\_Guide](http://wiki.eclipse.org/EGit/User_Guide)
- <https://schulung.t3isp.de/documents/pdfs/git/git-training.pdf>
- <https://git-scm.com/book/de/v2>
- <https://git-lfs.github.com/>
- [https://de.wikipedia.org/wiki/Liste\\_von\\_Git-GUIs](https://de.wikipedia.org/wiki/Liste_von_Git-GUIs)
- <https://git-scm.com/download/gui/windows>

#### 13. Off-Topic

- [Alternatives jira/confluence](#)

## SHA1 - checksums & backgrounds

- git extensively works with checksums in the background
- a checksum is a 40-char long hex-string ( a unique checksum of the data )
- every object gets a checksum
  - blogs
  - trees
  - commits
  - tags

## Lab 4: Find created object

```
# Works only within git bash or Linux
# Just for reference here !!
cd .git
cd objects
```

```
ls -la
# you will find a directory name with the
# first 2 chars of the object, e.g.
# drwxr-xr-x   3 jmetzger  staff  102 26 Okt 16:28 4f
```

```
# change into that directory
# Hint: Replace name with your directory name
cd 4f
# your object
ls -la
# 51d02eb27b6bdf1741ad48ccf6f7dc3326bbd2
```

```
# now let us see the content
# (taking the 4 letters of the object is sufficient)
git cat-file -p 4f51
my first line
```

```
# and the type of object
git cat-file -t 4f51
blob
```

## Git log - beautified (commandline)

- Why ?
  - For later usage, it will be easier to have a beautified log

```
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' --abbrev-commit"
```

## Lab 7: Setup beautified log (commandline)

```
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' --abbrev-commit"

# use beautified log
git lg
```

## Branches -> why ?

- important concept of git
- work on features easily
- independent from remote repository

## Create branches -> 1-step-version

- git checkout -b feature-4711

## Branch - which one is active ?

- git branch (star means active)

## change to another branch (commandline)

- `git checkout feature-4711`

## Lab 8: Create a new branch + work there (commandline)

```
git checkout -b feature-4711
# check which branch is active
git branch

# create a file feature-4711.txt
touch feature-4711.txt
# create a line of text into it
# with your favourite editor
# .e.g
# Line of code in 4711

# now commit the changes
git status
git add .
git commit -am "New feature-4711.txt"
git status
# do you notice, that the branch is ahead ?
git lg
```

## Merge changes -> merge

- You can merge, if you want to get changes from another branch
- A typical scenario would be:
  - You are in master
  - Checkout a feature branch
  - Work on a feature
  - Checkout master
  - merge changes from feature-branch
- You merge feature from another branch into your current branch
  - with -->
  - `git merge your-feature-branch`

## Lab 10: Exploring HEAD (commandline)

```
cd .git
cat HEAD
# ref: refs/heads/master
cat refs/heads/master
# f80891db4afa604b243bd06a5779fee88c3cad53
# compare this with the last commit - id
# What do you notice ?
git lg -1
```

## Replay changes in other branch + changes on current branch -> rebase (commandline)

- Change into branch (z.B.feature-4711)
- git rebase master
- -- how does it work ? ->
  - go back to the point where the branch was created
  - apply change from master
  - at the end apply changes from feature-4711
- -- ATTENTION ->
  - only do rebasing in your own "local" branch (NO ! collaborative branches)
  - never rebase, after push is done and branch is shared with others !

## Cancellation -> revert (commandline)

- Take back the last change \ But: you will have an additional log entry

```
# commandline
git revert
```

```
# Eclipse/EGit
1) Go to Log -> Right Click -> Show In -> History
2) Right Click on Commit (you want to revert) -> Revert
```

## From -> local repository -> to -> remote repository -> why ?

### Why ?

- data is saved outside of my system
- others can access it too (collaboration, e.g. on a feature)

## Publish the local changes remote

- (commandline) git push origin master
- (Eclipse/Egit): Right click -> Team -> Push to Upstream

## Tagging of a current version (locally)

- eclipse/EGit -> Team -> Advanced -> Tag

```
git tag -a v0.0.1 -m "Commit Message"
git push --tags
```

## Getting an old revision of a file

```
1) Rechte Maustaste
2) Replace With (on top of entry Team) Previous Revision/Commit
```

## Clone an online session into a new (none existant) directory ====

- git bash (on the desktop - right click)

- git clone <https://url> schulung2

## where is configuration saved ?

- on your local system the general configuration is saved in a file.
- Linux/Mac: ~/.gitconfig
- --
- example Mac: /Users/jmetzger/.gitconfig
- example Linux: /home/jmetzger/.gitconfig
- example Windows: C:\Users<user\_name>.gitconfig

## Commit - messages: what for and why should they be speaking ?

- other participants should be able to see changes based on commits
- good commit - message: reduces the time for search (for other participants) - more efficient
- search more easily and thoroughly (e.g. for later debugging)
- eventually included in the changelog and other tools

## Commit - message : structure

- line 1: short! summary only(!) chars and letters
- line 1: max. 50 chars
- line 1: best practice -> issue number/logical unit, then ":", then summary \ e.g. \ modulexy: Fixed problems with memory management
- line 2: EMPTY
- line 3: thorough description of commit
- line 3: max. 72 chars
- line 3: to structure: \*, -, # possible
- line 3: time: presence
- line 3: do not write, what do you did, but why.

## The cleanup: removal and untracking : git rm (commandline)

```
git rm
# removes the files locally (working directory)
# as well as for next staging.
# In the next commit the file will not be contained anymore.

git rm --cached
# if i only want to "untrack" a file (so it should be in the repo anymore),
# but want to keep it locally)
```

## Troubleshooting ssh -> repo (tortoise/openssh)

It is either possible to use openssh or plink.exe.  
Here are the most important settings for ssh.

- git bash: echo \$GIT\_SSH \ it should be: C:\Program Files\Git\usr\bin\ssh.exe here
- if not: export GIT\_SSH="C:\Program Files\Git\usr\bin\ssh.exe"
- in addition in tortoisegit under settings -> network the should be the following: \ C:\Program Files\Git\usr\bin\ssh.exe

## Workflows -> gitflow workflow

{{ :gitflow-workflow-4.png|}}

## Technical Background

### Projects with git

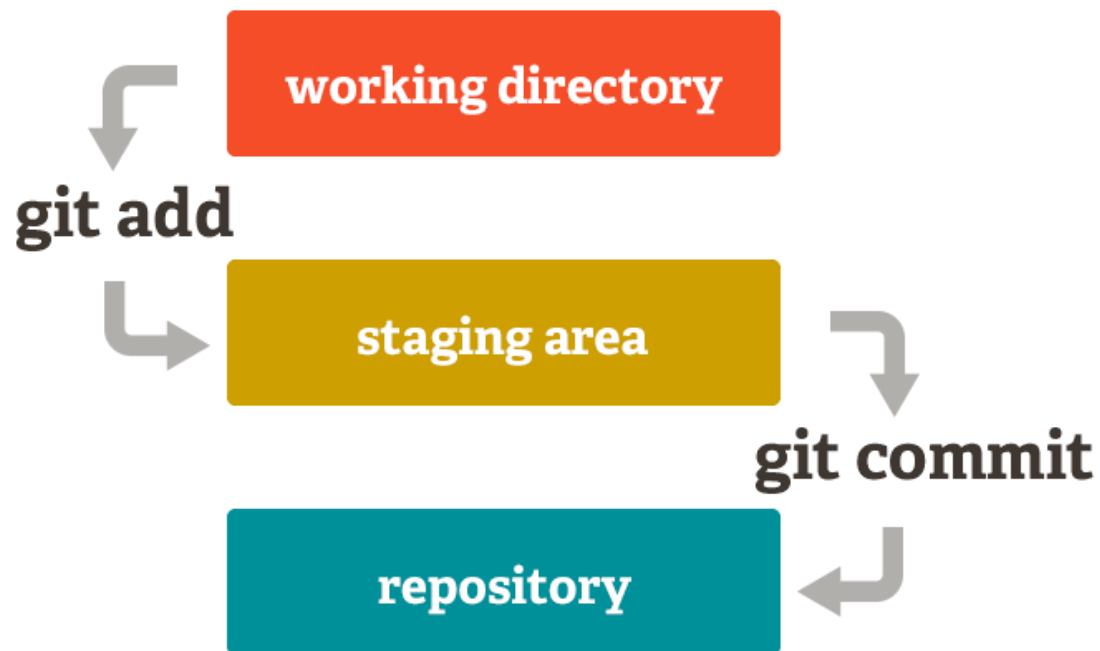
- Linux kernel project
- github
- ruby on rails
- phpmyadmin



### **How does it work?**

- git takes snapshots
- git works offline
  - (git saves the complete project including all versions)
- all files are saved in objects.

The flow of git



## Git and its objects

### Git and its objects

- To manage its data, git uses objects
- e.g. when you add a file with `git add filename` \ a new object is created
- There are 4 types of objects
  - blobs
  - trees
  - commits
  - tags

## 5-goldene-Regeln - Zerstörung

1. Kein `git commit --amend` auf bereits veröffentlicht (gepushed) commit.
2. Kein `git reset` vor bereits veröffentlichte (gepushed) commits  
(1234 (HEAD -letzter Commit) < 5412 (vö - HEAD~1 - vorletzte Commit) -> kein reset auf 1234)
- (3. Nie Struktur in 2 Branches (z.B. Master / Feature-Branch) vor Merge gleichzeitig ändern)
4. Mach niemals ein `git push --force` (JM sagt)
5. Kein Rebase auf bereits veröffentlichte commits (nach vö von Feature branchen)  
- ausser Feature-Branch kann online gelöscht und nochmal erstellt werden

## Commands - Commandline (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

* nonsense commit-message because 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 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

### Show branches

```
## locale branches + active branch (mit *)  
git branch  
## remote branches  
git branch -r  
## alle branches  
git branch -a
```

### 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
```

## **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

```
## 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 remote -v**

### **Examples**

```
## show configured remotes
git remote -v

## configure origin remote
## if it works -> no output
git remote add origin https://myurl.to/repo.git
```

## Eclipse/STS(Spring Tool Suite)

### Increasing Icon Size

- You might want to have bigger icon size (tested on Windows)
- Search for "eclipse.ini"
- Open in editor (e.g. Notepad++)
- Add the following 3 lines (at the end of the file):

```
-Dswt.enable.autoScale=true  
-Dswt.autoScale=200  
-Dswt.autoScale.method=nearest
```



## Installation of Eclipse

```
## go to https://www.eclipse.org/downloads
## Choose 32bit or 64bit
## in most cases 64bit (nowerdays)
## -> Downloads Eclipse installer

start eclipse installer
## -> choose "Eclipse IDE for Java Developers"

## Eclipse will be installed
```

## Installation of STS (Spring Tool Suite)

- We can find the version we need here:
  - <https://github.com/spring-projects/sts4/wiki/Previous-Versions>

```
## We will use the version 4.10.0
## in our case for mac
https://download.springsource.com/release/STS4/4.10.0.RELEASE/dist/e4.19/spring-tool-
suite-4-4.10.0.RELEASE-e4.19.0-macosx.cocoa.x86_64.dmg
```

## Prepare Perspective for good user experience

- It is important to have the right perspective/view

## Walkthrough

Todo 1:

Menu -> Window -> Show View -> Other -> Git -> Git Repositories

Todo 2:

Menu -> Window -> Show View -> Other -> Git -> Git Staging

Todo 3:

Menu -> Window -> Customize Perspective -> Action Set Availability -> Git && Apply & Close

Tab -> Menu Visibility -> Activate Git Checkbox (if not checked)

Tab -> Toolbar Visibility -> Activate Git Checkbox (if not checked)

Todo 4: Save perspective for further usage

Menu -> Windows -> Save Perspective As -> e.g. GIT/Workspace

## Create/Initialize Repository/Project

### Starting with a repo













- New Repo in eclipse
  - Go to Window -> Perspective -> Open Perspective -> Other -> Git
    - Use Icon 3 (from the left) of the git icons \ (Create a new git repository and add it to this view)
    - It suggests (the userdir)/git. Please add the name of your directory to your path (e.g. training)
    - Leave the box "create bare repository" UNCHECKED.
    - Click "Finish"
  - Go to new Project -> Java Project (if you are using java)
    - IMPORTANT: Uncheck "Use default location" (otherwise the repo will
- after that: All the intelligence and logic is within the subdirectory .git
- this means: folders/files are fully functional, also if the .git - folder is deleted

### Alternative: Starting with a project (recommended !)

- You can also start with a project
  - e.g. New -> Java Project ->
- After that make i a git-repo
  - Team -> Share
    - Do **Not** Check: Use or create repository in parent folder
    - Use the seperated git folder here
    - After doing that, all code will be moved to repo - folder
      - and here in the workspace
        - Example Structure
          - ~/git/RepoName/ <- Workspace
          - ~/git/RepoName/.git

## Eclipse Icons

These are the default decorations:

-  > Project [master]
-  > folder
-  tracked.txt
-  untracked.txt
-  ignored.txt
-  > dirty.txt
-  staged.txt
-  > partially-staged.txt
-  added.txt
-  removed.txt
-  > conflict.txt
-  assume-valid.txt

## Git-Commands in eGit

- [https://wiki.eclipse.org/EGit/Mapping\\_Git\\_Commands](https://wiki.eclipse.org/EGit/Mapping_Git_Commands)

git	eclipse/EGit
add:	"Add to Index" -> toolbar/menubar item to add all changes in selected files. "Team -> Add to Index" to add all changes in selected files. Drag-and-drop in "Git Staging" view Drag-and-drop in "Synchronization" view. "Compare With.." to stage or unstage line-by-line
annotate:	"Team -> Show in Annotations"
branch:	"Checkout"/"Switch To" toolbar/menubar item to list, create, rename and delete local branches and remote-references. toolbar of Commit views to create a branch from that commit right-click menu of commits listed in history
checkout:	"Checkout"/"Switch To" toolbar/menubar item to checkout a local branch. "Replace With..." to checkout a single file. right-click menu of commits listed in history or reflog right-click menu of branches in "Git Repositories" view toolbar of Commit views
cherry-pick:	right-click menu of commits listed in history. toolbar of Commit views
clone:	toolbar of Git Repositories" view
commit:	"Commit" toolbar/menubar item "Team -> Commit" "Commit" toolbar in "Git Staging" view
config:	"Properties" view Preferences: "Team -> Git" "Team -> Remote -> Configure..."
diff:	"Team -> Advanced -> Synchronize" to compare branches "Team -> Synchronize Workspace" to compare working tree to HEAD "Compare With" for specific files double-click items in either "Git Staging" changes windows
fetch:	"Fetch changes from upstream" toolbar/menubar "Team -> Fetch changes from upstream"
init:	toolbar of Git Repositories"
log:	"Team -> Show in history"
merge:	"Team -> Merge"
mv:	Implicit when file is renamed
pull:	"Pull" toolbar/menubar item "Team -> Pull"
push:	"Push to Upstream" toolbar/menubar item default push "Team -> Push to Upstream" \ ditto "Team -> Push..." for explicit push
rebase:	"Rebase" toolbar/menubar item "Team -> Rebase"
reflog:	"Git Reflog" view

remote:	"Team -> Remote" "Branches -> Remote Tracking" in "Git Repositories" view
reset:	"Reset" toolbar/menubar item to reset a branch "Team -> Reset" ditto
revert:	right-click menu of commits listed in history.
rm:	implicit when file is deleted
status:	implicit in decorations, "Git Staging View"
tag:	"Team -> Advanced -> Tag...", annotated tags only toolbar of Commit views, ditto

## Setup Identity

- [http://wiki.eclipse.org/EGit/User\\_Guide#Identifying\\_yourself](http://wiki.eclipse.org/EGit/User_Guide#Identifying_yourself)

Click Preferences > Team > Git > Configuration

Click New Entry and enter the key value pairs:

Key: user.name

Value: YourUsernameHere

And

Key: user.email

Value: YourEmailHere



## Staging Area

- In Eclipse you will see this information in the staging area:
  - New / Modified files (git status)

Right Click (Mouse) -> Team -> Commit

or:

Windows -> Show View -> Other -> Git -> Git Staging

## Log

```
Right Click -> Show In -> History  
or:  
Right Click -> Team -> Show In -> History
```

Alternative: (Recommended)

-> Simply do it from the staging area

## Commit

- 1) Right Click on Project
- 2) Team -> Commit

## Alternative

## Do it directly in the staging area

## Add

- Done within the Git Staging Area in EGit / Eclipse

Right Click (Mouse) -> Team -> Commit

or:

Windows -> Show View -> Other -> Git -> Git Staging

## Edit markdown in eclipse with plugin

- <https://marketplace.eclipse.org/content/markdown-text-editor>

## 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
```

## Features

### Partial Clone

### Prerequisites

```
at least git 2.22.0
```

### Command

```
git clone --filter=blob:none  
  
## blobs werden erst beim Checkout runtergeladen  
git checkout master
```

## Hooks

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

## GIT-Server

### GIT Server

#### Fully Fledged (können ziemlich viel) - Properitär

```
github
bitbucket
gitlab
```

#### Cloud and on premise

```
Installation im Netz oder in der Cloud
Aber: bitbucket (atlassian) verkaufen keine Lizenzen für on-premise (lokales Netz -
selber installiert)
```

#### gitlab

```
Es gibt eine Community Version (kostenlos) Anzahl Nutzer ?
```

#### github

```
Enterprise for own installation

https://github.com/organizations/enterprise_plan?
ref_cta=Start+a+free+trial&ref_loc=hero&ref_page=%2Fenterprise
```

#### bitbucket

```
bitbucket cloud - damit arbeiten wir
bitbucket server - lokale Installation (wird es nach 2021 nicht geben, bis ma 2024)
bitbucket data center
```

#### Kleinere Server (können fast nix)

```
git-server (direkt)
https://git-scm.com/book/de/v2/Git-auf-dem-Server-Einrichten-des-Servers

gitosis
https://www.admin-magazin.de/Das-Heft/2012/02/Git-Server-selbst-
installieren/(offset)/2#:~:text=Gitosis,die%20entsprechenden%20Skripte%20aufgerufen%20we

web-git
```

#### Freie Alternativen (online - cloud)



codeberg.org

### **Freie Alternativen (offline ;o) - on premise)**

<https://gitea.io/en-us/> (codeberg basiert darauf)

### **Welchen nehme ich (Vergleich bitbucket / gitlab)**

- <https://www.atlassian.com/de/software/bitbucket/comparison/bitbucket-vs-gitlab>

### **Welchen nehme ich (Vergleich bitbucket / github)**

- <https://gist.github.com/juderosen/8410710>

# Golden

## 5-goldene-Regeln

1. Kein `git commit --amend` auf bereits veröffentlicht (gepushed) commit.
2. Kein `git reset` vor bereits veröffentlichte (gepushed) commits  
(1234 (HEAD -letzter Commit) < 5412 (vö - HEAD~1 - vorletzte Commit) -> kein reset auf 1234)
3. Nie Struktur in 2 Branches (z.B. Master / Feature-Branch) vor Merge gleichzeitig ändern)
4. Mach niemals ein `git push --force` (JM sagt)
5. Kein Rebase auf bereits veröffentlichte commits (nach vö von Feature branchen)  
- ausser Feature-Branch kann online gelöscht und nochmal erstellt werden

## Tipps & Tricks (git / git bash)

### Konfigurationseinstellung löschen

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

## Anderen Editor verwenden

```
git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst  
-notabbar -nosession -noPlugin"
```

## Platz sparen mit dem shallow clone + neu pushen

### Walkthrough (nicht pushbar)

```
## ohne Einschränkung 1428 objekte aktuell (stand 09.09.2021)
git clone https://github.com/jmetzger/mariadb-fuer-entwickler.git

### Clone only last 501 commits
git clone --depth=501 https://github.com/jmetzger/mariadb-fuer-entwickler.git mfe
cd mfe

### Es steht der Hinweis grafts im letzten Logeintrag.
### Der letzte Logeintrag (also das erste commit hat ein parent),
### dadurch kann es nicht in ein neues Repo gepushed werden.
```

### Walkthrough (pushbar)

```
### Nur 201 commits abholen damit das handling einfacher ist.
git clone --depth=501 https://github.com/jmetzger/training-git.git gt

### Jetzt Historie neu aufbauen
cd gt
### sha - id des ersten commits (letzter log eintrag finden)
### z.B. cc43
### tree rausfiltern und auf dieser Basis einen neuen Initial-Commit ohne eltern
erstellen.
echo "Start of truncated history" | git commit-tree cc43^{tree}
### es erscheint die neue commit-id z.B. 56c3.....

### Diese nehmen wir um darauf die restliche Historie draufzuflanschen
git rebase --onto 56c3 cc43

### historie ist fertig

### Jetzt remote umbiegen
git remote set-url origin https://url_zum/neuen_repo.git

### bzw. master
git push origin main
```

### Branch aus bestehendem Branch neu erstellen (alles auf Anfang!)

```
git checkout --orphan new_branch
git status
git add .
git commit -am "New Initial release"
git checkout main
```

## Arbeiten mit git replace

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

## ssl Verifizierung ausschalten (quick & dirty)

```
## Wenn eine Fehler auftritt, wie z.B. beim Pushen
fatal: unable to access 'https://bitbucket.org/xy/xyz.git/': SSL certificate problem:
unable to get local issuer certificate

git config http.sslVerify false

oder: für alle repos
git config --global http.sslVerify false
```

# GIT Mergetools

## git mergetools

### Welche Tools sind auf dem System vorhanden

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

### Meld (Windows)

- <https://meldmerge.org/>

### Meld - Configuration in Git for Windwos (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"
## do not create an .orig - file before merge
git config --global mergetool.keepBackup false
```

### TortoiseGitMerge - Configuration

```
## If you have tortoisegit installed on your system, you can use tortoisegitmerge for
git for windows as well
## Do not keep orig-file
git config --global merge.tool tortoisegit
git config --global mergetool.keepBackup false
```

### How to use it

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



## GIT-Guis

### Exercises

#### Create simple conflict on commandline

```
## if todo.txt does not exist, create it after creating new branch
1. neuen branch: feature/4920 - todo.txt Zeile 1 und Zeile4
2. add + commit
3. git checkout master
4. in master -> todo.txt zeile 1 und zeile 4 ändern
5. add + commit
6. git merge feature/4920
```

## Conflict merge-request

Schritt 1:

Online die Datei src/README.txt im master

-> ändern Zeile 1, Zeile 4, Zeile 8 (wenn nicht vorhanden anlegen) -> im master

Schritt2:

1. lokal neuen Branch feature/7001 erstellen und checkout

2. im branch src/README.txt anlegen (wenn nicht vorhanden)

und Zeile 1 , Zeile4, Zeile 8 etwas reinschreiben

3. add + commit

4. git push -u origin feature/7001

5.. merge-request

-> feature/7001 -> master

6.. ihr versucht zu mergen

## Documentation

### GIT pdf

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

## Off-Topic

### Alternatives jira/confluence

#### Jira

```
Trac
Roundup
OTRS
Taiga
Redmine (speziell für Software)
Open Project

-> Redmine
```

### GIT-Server

```
gitea
```

### Wikis

```
dokuwiki
-> Migration
```