

Ordnerpaket-Notiz

Jan Unger

5. April 2020



Inhalt

| | | |
|----------|--|----------|
| 1 | Ordnerpaket-Notiz | 1 |
| 1.1 | Abbildungen | 1 |
| 1.2 | Quellcode | 5 |
| 1.3 | Git - Schnellstart | 6 |
| 1.3.1 | Git konfigurieren | 6 |
| 1.3.2 | Projekt mit GitHub einrichten | 6 |
| 1.3.3 | Projekt von GitHub downloaden | 7 |
| 1.3.4 | Projekt lokal einrichten | 7 |
| 1.3.5 | Git Workflow | 8 |
| 1.3.6 | git log - history | 9 |
| 1.3.7 | Branch neu erstellen - PROJEKT bearbeiten | 9 |
| 1.3.8 | git tag - Version erstellen | 10 |
| 1.3.9 | git blame - Wer hat was und wann geändert? | 10 |
| 1.3.10 | Datei umbenennen o. löschen - .git o. Repository löschen | 10 |
| 1.3.11 | Versionskonflikt lösen | 10 |
| 1.3.12 | Lokales Wiederherstellen | 11 |
| 1.3.13 | Repository Wiederherstellen | 12 |
| 1.3.14 | Git Backup - Repository ohne einen Workspace | 13 |
| 1.3.15 | lokales Repository clonen | 14 |
| 1.3.16 | Git Workflow - Bereiche | 14 |
| 1.3.17 | Wiederherstellen | 15 |
| 1.3.18 | Repositorys auf unterschiede prüfen | 17 |
| 1.4 | Markdown | 20 |
| 1.4.1 | Überschriften | 20 |
| 1.4.2 | Code | 20 |
| 1.4.3 | Quellenangabe | 20 |
| 1.4.4 | Listen | 20 |
| 1.4.5 | Links | 21 |
| 1.4.6 | Absätze | 21 |
| 1.4.7 | Texthervorhebung | 22 |
| 1.4.8 | Bild | 22 |
| 1.4.9 | Tabelle | 22 |

| | | |
|--------|--------------------------------------|----|
| 1.4.10 | Mathe | 23 |
| 1.5 | Notiz | 25 |
| 1.6 | Readme | 26 |
| 1.6.1 | Inhalt - Ordnerpaket-Notiz | 26 |
| 1.6.2 | Erste Schritte | 26 |
| 1.6.3 | Git | 27 |

1Ordnerpaket-Notiz

1.1 Abbildungen

»logo« (vgl. Abb. [1.1](#)).



Abb. 1.1: logo

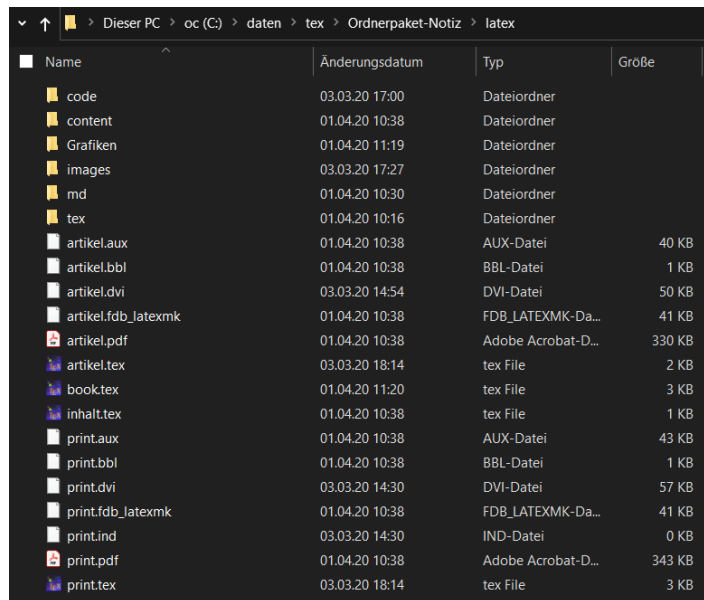
»logo-negativ« (vgl. Abb. [1.2](#)).



Abb. 1.2: logo-negativ

»Ordner-latex« (vgl. Abb. [1.3](#)).

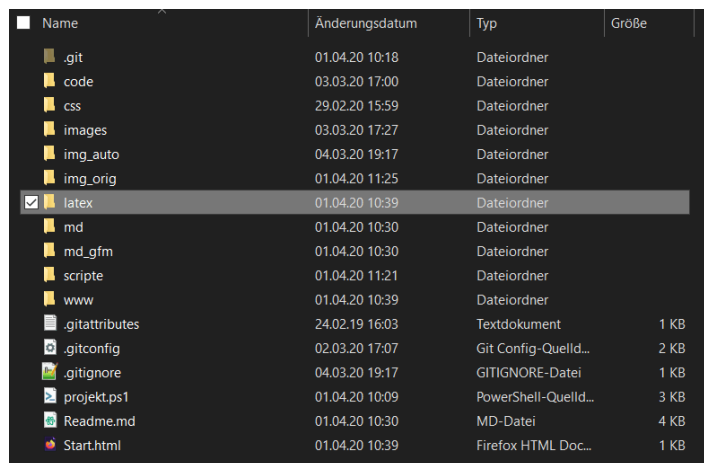
1 Ordnerpaket-Notiz



| Name | Änderungsdatum | Typ | Größe |
|---------------------|----------------|--------------------|--------|
| code | 03.03.20 17:00 | Dateiordner | |
| content | 01.04.20 10:38 | Dateiordner | |
| Grafiken | 01.04.20 11:19 | Dateiordner | |
| images | 03.03.20 17:27 | Dateiordner | |
| md | 01.04.20 10:30 | Dateiordner | |
| tex | 01.04.20 10:16 | Dateiordner | |
| artikel.aux | 01.04.20 10:38 | AUX-Datei | 40 KB |
| artikel.bbl | 01.04.20 10:38 | BBL-Datei | 1 KB |
| artikel.dvi | 03.03.20 14:54 | DVI-Datei | 50 KB |
| artikel.fdb_latexmk | 01.04.20 10:38 | FDB_LATEXMK-Da... | 41 KB |
| artikel.pdf | 01.04.20 10:38 | Adobe Acrobat-D... | 330 KB |
| artikel.tex | 03.03.20 18:14 | tex File | 2 KB |
| book.tex | 01.04.20 11:20 | tex File | 3 KB |
| inhalt.tex | 01.04.20 10:38 | tex File | 1 KB |
| print.aux | 01.04.20 10:38 | AUX-Datei | 43 KB |
| print.bbl | 01.04.20 10:38 | BBL-Datei | 1 KB |
| print.dvi | 03.03.20 14:30 | DVI-Datei | 57 KB |
| print.fdb_latexmk | 01.04.20 10:38 | FDB_LATEXMK-Da... | 41 KB |
| print.ind | 03.03.20 14:30 | IND-Datei | 0 KB |
| print.pdf | 01.04.20 10:38 | Adobe Acrobat-D... | 343 KB |
| print.tex | 03.03.20 18:14 | tex File | 3 KB |

Abb. 1.3: Ordner-latex

»Ordner-struktur« (vgl. Abb. 1.4).



| Name | Änderungsdatum | Typ | Größe |
|----------------|----------------|---------------------|-------|
| .git | 01.04.20 10:18 | Dateiordner | |
| code | 03.03.20 17:00 | Dateiordner | |
| css | 29.02.20 15:59 | Dateiordner | |
| images | 03.03.20 17:27 | Dateiordner | |
| img_auto | 04.03.20 19:17 | Dateiordner | |
| img_orig | 01.04.20 11:25 | Dateiordner | |
| latex | 01.04.20 10:39 | Dateiordner | |
| md | 01.04.20 10:30 | Dateiordner | |
| md_gfm | 01.04.20 10:30 | Dateiordner | |
| scripte | 01.04.20 11:21 | Dateiordner | |
| www | 01.04.20 10:39 | Dateiordner | |
| .gitattributes | 24.02.19 16:03 | Textdokument | 1 KB |
| .gitconfig | 02.03.20 17:07 | Git Config-Quell... | 2 KB |
| .gitignore | 04.03.20 19:17 | GITIGNORE-Datei | 1 KB |
| projekt.ps1 | 01.04.20 10:09 | PowerShell-Quell... | 3 KB |
| Readme.md | 01.04.20 10:30 | MD-Datei | 4 KB |
| Start.html | 01.04.20 10:39 | Firefox HTML Doc... | 1 KB |

Abb. 1.4: Ordner-struktur

»Sport-Kraft« (vgl. Abb. 1.5).



Abb. 1.5: Sport-Kraft

»Sport-Winter« (vgl. Abb. 1.6).



Abb. 1.6: Sport-Winter

1.2 Quellcode

Programm »hallo.c« (vgl. Quelltext [1.1](#)).

Quelltext 1.1: hallo.c

```
// ju 9-Jun-19 hallo.c
#include <stdio.h>
int main(void){
    printf("Hallo Welt!\n");
    return 0;
}
```

Programm »hallowelt.c« (vgl. Quelltext [1.2](#)).

Quelltext 1.2: hallowelt.c

```
// ju 3-Mrz-20 hallowelt.c
#include <stdio.h>
int main(void){
    printf("Hallo Welt!\n");
    return 0;
}
```


1.3 Git - Schnellstart

Befehle

- Git: download <https://git-scm.com/download/win>
- GitHub: Projekt anlegen <https://github.com/ju1-eu/>
- git clone »Repository_name«
- git add »file« oder git add .
- git commit -a oder git commit
- git push u. git pull
- git status
- git log oder git lg
- git stash
- git reset

1.3.1 Git konfigurieren

```
# Git konfigurieren
#-----
git version

# gitconfig anpassen
git config --global user.name "jan_wlap"
git config --global user.email "esel573@gmail.com"

# gitconfig ansehen
git config --global --list
# cd C:\Users\jan\
  .gitconfig
  .gitattributes
  .gitignore
```

1.3.2 Projekt mit GitHub einrichten

Auf GitHub ein Repository erstellen: <https://github.com/ju1-eu/>

- **Repository_name:** »prj-dummy«
- **Clone with HTTPS:** <https://github.com/ju1-eu/prj-dummy.git>
- weiter mit »Projekt von GitHub downloaden«
- oder
- weiter mit »Projekt lokal einrichten«

1.3.3 Projekt von GitHub downloaden

pwd

```
$THEMA="prj-dummy"
rm $THEMA -Recurse -Force
git clone https://github.com/ju1-eu/$THEMA.git
#git clone https://github.com/ju1-eu/$THEMA.git prj1

# weiter mit ...
# Git Workflow / PROJEKT bearbeiten
```

1.3.4 Projekt lokal einrichten

pwd

```
# lokales Repository: master
$THEMA="prj-dummy" # Repository_name auf Github erstellen
#rm $THEMA -Recurse -Force
mkdir $THEMA
cd $THEMA
# Repository anlegen
git init # rm -rf .git oder rm .git -Recurse -Force
echo "# prj-dummy" > README.md
git add .
git commit -m"init"
git status
git remote add origin https://github.com/ju1-eu/$THEMA.git
git push --set-upstream origin master

# weiter mit ...
# Git Workflow / PROJEKT bearbeiten
```

1.3.5 Git Workflow

1.3.5.1 Arbeitsverzeichnis

```
pwd
# /c/daten/projekte/git/git-kurs/prj
# C:\daten\projekte\git\git-kurs\prj
```

1.3.5.2 PROJEKT bearbeiten

```
git status

# README.md erstellen
echo "# prj-dummy" > README.md
echo "<!-- update: 20-Dez-19 -->" >> README.md
cat README.md

# .gitignore erstellen
echo "# ju update: 20-Dez-19 .gitignore" > .gitignore
echo "!.gitignore" >> .gitignore
cat .gitignore
```

1.3.5.3 git add - Arbeitsverzeichnis => Stagingbereich

```
git add .
git add README.md
```

1.3.5.4 git commit - Stagingbereich => Repository

```
git commit -m"projekt init"
git commit
git commit -a
```

1.3.5.5 git push - Repository => Github Repository

```
# Voraussetzung
$THEMA="prj-dummy" # Repository_name auf Github erstellen
git remote add origin https://github.com/ju1-eu/$THEMA.git
git push --set-upstream origin master

git status
git push
git pull
```

1.3.5.6 git status - sauberes Arbeitsverzeichnis prüfen!

```
git status
On branch master
Your branch is up to date with 'origin/master'.
```

```
nothing to commit, working tree clean
```

1.3.6 git log - history

```
git status
git log
git log --abbrev-commit --pretty=oneline --graph
git lg
```

1.3.7 Branch neu erstellen - PROJEKT bearbeiten

```
# sauberes Arbeitsverzeichnis prüfen!
git checkout master
git status
# wenn lokale Änderungen vorliegen, wird der Merge abgebrochen
git pull --ff-only
```

```
git branch
# Branch neu erstellen
git checkout -b feature/a1
# Feature-Branch zentral sichern
git push --set-upstream origin feature/a1
# Änderungen können zukünftig gesichert werden
git push
```

```
# PROJEKT bearbeiten
code README.md
git add .
git commit
```

```
# Branch wechseln
git status
git branch
git checkout master
```

```
# Branch zusammenführen
git merge feature/a1
```

```
git branch
# nutzlosen Branch löschen
git push -d origin feature/a1
```

```
git branch -d feature/a1

# sauberes Arbeitsverzeichnis prüfen!
git status
```

1.3.8 git tag - Version erstellen

```
# sauberes Arbeitsverzeichnis prüfen!
git status

git tag
git tag -a v1.0
# stabile Version

git checkout v1.0
git checkout master
```

1.3.9 git blame - Wer hat was und wann geändert?

```
git blame README.md
```

1.3.10 Datei umbenennen o. löschen - .git o. Repository löschen

```
git mv file_alt file_neu
git commit -am"Datei umbenannt"

git rm file_neu
git commit -am"Datei gelöscht"
git status

# .git löschen
rm -rf .git
rm .git -Recurse -Force

# Repository löschen
git remote -v
git remote rm backupHD
git remote rm origin
```

1.3.11 Versionskonflikt lösen

```
# sauberes Arbeitsverzeichnis prüfen!
git status

git branch -a
```

```
# Voraussetzung: Branch muss vorhanden sein
# siehe Branch neu erstellen
git checkout feature/a1

# PROJEKT bearbeiten auf feature/a1 Branch
code README.md
# file: README.md bearbeiten
git commit -am"README.md bearbeitet"
git push
git diff master feature/a1

git checkout master

# PROJEKT bearbeiten auf master Branch
code README.md
# file: README.md bearbeiten
git commit -am"README.md bearbeitet master"
git push
git status

# zusammenführen mit master-Branch
git merge feature/a1
# Fehler, weil auf unterschiedlichen Branches gearbeitet wurde
#-----
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
#-----
# Konflikt bearbeiten
code README.md
# file bereinigen: <<<<<<, ===== und >>>>>>

git commit -am"Konflikt behoben"
git push

git branch -a
git push -d origin feature/a1
git branch -d feature/a1

# sauberes Arbeitsverzeichnis prüfen!
git status
```

1.3.12 Lokales Wiederherstellen

```
# Änderungen zwischenspeichern
git stash --include-untracked
# Letzte gespeicherte Änderungen zurückholen
git stash pop
```

pwd

```
# PROJEKT bearbeiten
echo "neu" > file.txt
git add .
git commit -m"neu"
echo "Fehler" >> file.txt
# Achtung: noch kein git commit!
git status
```

```
# Möglichkeit 1
git log
cat .\file.txt
git checkout HEAD file.txt
cat .\file.txt
git status
```

```
# Möglichkeit 2
git log
cat .\file.txt
git reset --hard HEAD
cat .\file.txt
git status
```

1.3.13 Repository Wiederherstellen

pwd

```
# PROJEKT bearbeiten
echo "Fehler 2" >> file.txt
git commit -am"Fehler provoziert"
# Achtung: noch kein git push!
git status
```

```
# Möglichkeit 1
git log
git reset --hard d446456
git log
git status
```

```
# Möglichkeit 2
git log
git revert c3b6fab
git log
git status
```

```
git push
git status
```

1.3.14 Git Backup - Repository ohne einen Workspace

Backup auf USB-Stick

```
pwd

# backup Repository: backup_wlap/master
# anpassen
$PFAD="E:/backup-Repos/notizenWin10"
$THEMA="prj-dummy"
$REPO="backup_wlap"

#cd $THEMA
git clone --no-hardlinks --bare . $PFAD/$THEMA.git
git remote add $REPO $PFAD/$THEMA.git
# gelegentlich sichern
git status
git push --all $REPO

# löschen
rm -rf $PFAD/$THEMA
rm $PFAD/$THEMA -Recurse -Force
```

Backup auf Raspberry Pi

```
pwd

# backup Repository: backup_rpi4/master
# anpassen
$PFAD="//RPI4\usbstick\backup-Repos\notizenWin10"
$THEMA="prj-dummy"
$REPO="backup_rpi4"

#cd $THEMA
git clone --no-hardlinks --bare . $PFAD/$THEMA.git
git remote add $REPO $PFAD/$THEMA.git
# gelegentlich sichern
```



```
git status
git push --all $REPO

# löschen
rm -rf $PFAD/$THEMA
rm $PFAD/$THEMA -Recurse -Force
```

1.3.15 lokales Repository clonen

```
pwd
cd C:\daten\projekte\git\git-kurs\prj

# löschen
rm -rf $THEMA
rm $THEMA -Recurse -Force

$PFAD="E:/backup-Repos/notizenWin10"
$THEMA="prj-dummy"
$REPO="backup_wlap"

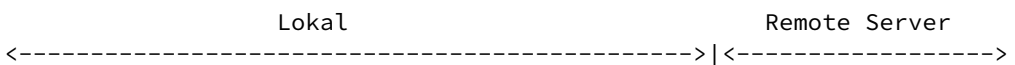
git clone $PFAD/$THEMA.git
cd $THEMA

# oder

$PFAD="\\RPI4\usbstick\backup-Repos\nutzenWin10"
$THEMA="prj-dummy"
$REPO="backup_rpi4"
git clone $PFAD/$THEMA.git
cd $THEMA
```

1.3.16 Git Workflow - Bereiche

- Datei bearbeiten
- Datei hinzufügen
- Datei Umbenennen
- Datei löschen



Arbeitsverzeichnis | Stagingbereich | Repository | Github

Repository

bearbeiten

git add

git mv

git rm

git commit

git push

<-----
>|<----->

1.3.17 Wiederherstellen

1.3.17.1 Ordner für Experimente erstellen - löschen

pwd

mkdir repoWork

mkdir repoNeu

mkdir repoAlt

löschen

rm repoWork -Recurse -Force

rm repoNeu -Recurse -Force

rm repoAlt -Recurse -Force

1.3.17.2 bestehendes Repository clonen

cd prj-dummy # Repository

git clone . ../repoWork

1.3.17.3 Arbeitsverzeichnis bearbeiten

bearbeiten 1

cd ../repoWork

code test.md

file

basis

git versionieren

git add .

git commit -a

git status

bearbeiten 2

```
code test.md
# file
Basis
Version2
```

```
# git versionieren
git commit -a
git status
```

bearbeiten 3

```
code test.md
# file
Basis
Version2
Version3
```

```
# git versionieren
git commit -a
git status
git log --graph --oneline
```

1.3.17.4 Wiederherstellen: Repository in ein temp. Verzeichnis klonen

```
cd repoWork
git clone . ../repoNeu
git clone . ../repoAlt
```

1.3.17.5 Wechsel auf den gewünschten Git-Branch

```
cd ../repoNeu/
git stash
git log --graph --oneline
* 0aa9744 (HEAD -> master, origin/master, origin/HEAD) version3
* 44aba18 version2
* 81ac502 basis
# wechsel auf version2
git reset --hard 44aba18
```

1.3.17.6 Verschiebe .git in den Workspace der alten Versionsverwaltung

```
# win10
Suchen: git bash
```

```
cd /c/daten/projekte/git/git-kurs/prj/repoNeu
```

```
git archive master | tar -x -C ../repoAlt/  
#git archive master | gzip > ../latest.tgz
```

1.3.17.7 Ergebnis prüfen

```
cd ..  
kdiff3 repoAlt/ repoNeu/
```

1.3.18 Repositories auf Unterschiede prüfen

1.3.18.1 Ordner für Experimente erstellen - löschen

```
pwd  
  
mkdir repoLokal  
mkdir repoGithub  
mkdir repoBackup_wlap  
mkdir repoBackup_rpi4  
  
# löschen  
rm repoLokal -Recurse -Force  
rm repoGithub -Recurse -Force  
rm repoBackup_wlap -Recurse -Force  
rm repoBackup_rpi4 -Recurse -Force
```

1.3.18.2 bestehendes Repository clonen

```
cd prj-dummy # Repository  
git status  
git lg  
git push  
  
# Git Backup - Repository ohne einen Workspace  
$REPO="backup_rpi4"  
git push --all $REPO  
  
$REPO="backup_wlap"  
git push --all $REPO  
  
git clone . ../repoWork
```

1.3.18.3 lokales Repository

HEAD -> master

```
cd repoWork
# repository clonen
git clone . ../repoLokal

# backup
cd ../repoLokal
#ls . | Compress-Archive -Update -dest ../repoLokal.zip
$verz="repoLokal"
$ID=$(git rev-parse --short HEAD) # git commit (hashwert)
$timestamp = Get-Date -Format 'd-MMM-y' # 10-Jun-19
$archiv = "$verz-$ID-$timestamp"
ls . | Compress-Archive -dest ../$archiv.zip
cd ..
```

1.3.18.4 Github Repository

origin/master

```
cd repoGithub
# repository clonen
$THEMA="prj-dummy"
git clone https://github.com/jul-eu/$THEMA.git .

# backup
#ls . | Compress-Archive -Update -dest ../repoLokal.zip
$verz="repoGithub"
$ID=$(git rev-parse --short HEAD) # git commit (hashwert)
$timestamp = Get-Date -Format 'd-MMM-y' # 10-Jun-19
$archiv = "$verz-$ID-$timestamp"
ls . | Compress-Archive -dest ../$archiv.zip
cd ..
```

1.3.18.5 lokales backup Repository

backup_wlap/master

```
cd repoBackup_wlap
# repository clonen
$PFAD="E:/backup-Repos/notizenWin10"
$THEMA="prj-dummy"
$REPO="backup_wlap"

git clone $PFAD/$THEMA.git .
```

```
# backup
#ls . | Compress-Archive -Update -dest ../repoBackup_wlap.zip
$verz="repoBackup_wlap"
$ID=$(git rev-parse --short HEAD) # git commit (hashwert)
$timestamp = Get-Date -Format 'd-MMM-y' # 10-Jun-19
$archiv = "$verz-$ID-$timestamp"
ls . | Compress-Archive -dest ../$archiv.zip
cd ..
```

backup_rpi4/master

```
cd repoBackup_rpi4
# repository clonen
$PFAD="\\RPI4\usbstick\backup-Repos\notizenWin10"
$THEMA="prj-dummy"
$REPO="backup_rpi4"
```

```
git clone $PFAD/$THEMA.git .
```

```
# backup
#ls . | Compress-Archive -Update -dest ../repoBackup_rpi4.zip
$verz="repoBackup_rpi4"
$ID=$(git rev-parse --short HEAD) # git commit (hashwert)
$timestamp = Get-Date -Format 'd-MMM-y' # 10-Jun-19
$archiv = "$verz-$ID-$timestamp"
ls . | Compress-Archive -dest ../$archiv.zip
cd ..
```

1.3.18.6 Ergebnis prüfen

```
pwd
```

```
# verzeichnisse vergleichen
kdiff3 repoLokal/ repoGithub/
kdiff3 repoLokal/ repoBackup_rpi4/ repoBackup_wlap/
```

```
# files vergleichen
kdiff3 repoLokal/test.md repoGithub/test.md
kdiff3 repoLokal/test.md repoBackup_rpi4/test.md
      repoBackup_wlap/test.md
```

1.4 Markdown

1.4.1 Überschriften

```
# Überschrift 1
## Überschrift 2
### Überschrift 3
```

1.4.2 Code

Quellcode

```
#include <stdio.h>
int main(void) {
    printf("Hallo Welt!\n");
    return 0;
}
```

1.4.3 Quellenangabe

Zitat: vgl. u.

```
\cite{monk_action_buch:2016}
\cite{kofler_linux:2017}
\footnote{\url{https://de.wikipedia.org/wiki/LaTeX}}.
```

1.4.4 Listen

ungeordnete Liste

- a
- b
 - bb
- c
 - a
 - b
 - * bb
 - c

Sortierte Liste

1. eins
 2. zwei
 3. drei
1. eins
 2. zwei
 3. drei

Sortierte Liste

1. a
 2. b
 3. c
1. a
 2. b
 3. c

1.4.5 Links

<https://google.de> oder Google

<<https://google.de>>

[Google](<https://google.de>)

1.4.6 Absätze

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn« ? Kjift - mitnichten! Ein Blindtext bietet mir wichtige Informationen.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist

das wirklich so? Ist es gleichgültig, ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn« ? Kjift - mitnichten! Ein Blindtext bietet mir wichtige Informationen.

1.4.7 Texthervorhebung

fett *kursiv* »Anführungsstriche«

****fett****

kursiv

"Anführungsstriche"

1.4.8 Bild

Bilder in eps o. pdf speichern, empfehlenswert für LaTeX.

Für das Web webp, png, jpg o. svg.



Abb. 1.7: Sport-Winter: erfolgt auto. eps -> pdf

![Sport-Winter](images/Sport-Winter.eps)

1.4.9 Tabelle

Tabellengenerator aus *.csv files

| **Nr.** | **Begriffe** | **Erklärung** |
|----------------|---------------------|----------------------|
| ---- | :----- | :----- |
| 1 | a1 | a2 |

| Nr. | Begriffe | Erklärung |
|-----|----------|-----------|
| 1 | a1 | a2 |
| 2 | b1 | b2 |
| 3 | c1 | c2 |

| | | | |
|---|----|----|--|
| 2 | b1 | b2 | |
| 3 | c1 | c2 | |

1.4.10 Mathe

$$[V] = [\Omega] \cdot [A] \text{ o. } U = R \cdot I$$

$$[V] = [\Omega] \cdot [A] \text{ o. } U = R \cdot I$$

$$R = \frac{U}{I}$$

$$R = \frac{U}{I}$$



Abb. 1.8: Logo: pdf

1.5 Notiz

1.6 Readme

(c) 2020 Jan Unger

<https://bw-ju.de>

1.6.1 Inhalt - Ordnerpaket-Notiz

| | |
|----------------|--|
| code/* | Quellcode |
| css/* | Webdesign anpassen |
| images/ | Bilder optimiert |
| img_auto/ | temp |
| img_orig/ | Bilder in Original: min. 2x *.jpg o. *.png |
| latex/ | LaTeX files -> pdfs |
| Grafiken/ | logo u. titelbild |
| content/ | präambel, metadata, header, literatur.bib |
| md/ | Notizen erstellen in Markdown |
| md_gfm/ | GitHub Flavored Markdown |
| scripte/ | PS-Scripte werden von projekt.ps1 aufgerufen |
| www/ | CMS Wordpress u. HTML5 files |
| .gitattributes | Git |
| .gitconfig | |
| .gitignore | |

| | |
|-------------|--|
| projekt.ps1 | PS-Script ausführen => erstellt pdfs u. HTML, Backup, Git, opti. Bilder ... |
| Readme.md | lesen! |
| Start.html | Projekt - Website öffnen |

```
# Backup files
cd ..
DATUM_Thema-Notiz_vVERSION.zip
Thema-Notiz.zip
git_log.txt
```

1.6.2 Erste Schritte

```
# Projektordner öffnen
git clone https://github.com/jul-eu/Ordnerpaket-Notiz.git notiz
cd notiz
```

»Readme.txt« **lesen**

Thema im Script projekt.ps1 anpassen u. im Ordner

```
scripte/"git.ps1" u. "backup.ps1"
latex/content/"metadata.tex", "zusammenfassung.tex", "literatur.bib"
latex/Grafiken/"titelbild.pdf"
```

Notizen in Markdown erstellen: min. 2x! md/*.md

```
# PS-Script ausführen
```

```
PS >_ projekt.ps1
```

```
# Projekt - Website öffnen
```

```
Start.html
```

1.6.3 Git

1.6.3.1 Repository clonen

```
# Github
```

```
$THEMA      = "Ordnerpaket-Notiz" # Repository
```

```
$ADRESSE     = "https://github.com/ju1-eu"
```

```
git clone $ADRESSE/${THEMA}.git
```

```
# USB
```

```
$USB        = "E:\repos\notizenWin10"
```

```
$THEMA      = "Ordnerpaket-Notiz" # Repository
```

```
git clone $USB/${THEMA}.git
```

```
# RPI
```

```
$RPI        = "\\RPI4\nas\repos\notizenWin10"
```

```
$THEMA      = "Ordnerpaket-Notiz" # Repository
```

```
git clone $RPI/${THEMA}.git
```

1.6.3.2 Repository neu anlegen

```
# Github
```

```
$THEMA      = "Ordnerpaket-Notiz" # Repository
```

```
$ADRESSE     = "https://github.com/ju1-eu"
```

```
git remote add origin $ADRESSE/${THEMA}.git
```

```
git push --set-upstream origin master
```

```
# backupUSB
```

```
$USB        = "E:\repos\notizenWin10"
```

```
$THEMA      = "Ordnerpaket-Notiz" # Repository
```

```
$LESEZ      = "backupUSB"
```

```
git clone --no-hardlinks --bare . $USB/${THEMA}.git
```

```
git remote add $LESEZ $USB/${THEMA}.git
```

```
git push --all $LESEZ
```

```
# backupRPI
$RPI = "\\RPI4\nas\repos\notizenWin10"
$THEMA = "Ordnerpaket-Notiz" # Repository
$LESEZ = "backupRPI"
git clone --no-hardlinks --bare . $RPI/${THEMA}.git
git remote add $LESEZ $RPI/${THEMA}.git
git push --all $LESEZ
```

1.6.3.3 Versionsverwaltung git

sichern - löschen - umbenennen

```
cd notiz
#git init # Repository neu erstellen
git add .
#git commit -am "Projekt start"
git commit -a
# letzten Commit rückgängig
git commit --amend
```

```
# sichern
git push # github
git push --all backupUSB
git push --all backupRPI
```

```
# löschen
git rm datei o. ordner
git rm -rf ordner
git rm ordner -Recurse -Force
git remote -v
git remote rm backupHD
```

```
# umbenennen
git mv datei datei_neu
```