Projekt

ordnerpaket-notizenDummy-v02

Mitschrift

von
Jan Unger

Wuppertal

21. Oktober 2018

Inhaltsverzeichnis

1. Kapitel

1.1 Readme

% ju – https://bw1.eu – 10-Okt-18 – Readme.md

1.1.1 Hinweis

Projekt getestet unter Win10

1.1.2 Software

Pandoc: https://pandoc.org/installing.html
Latex: https://www.tug.org/texlive/acquire-netinstall.html

```
# Shell: TeXlive update
tlmgr update --all
```

Editor: https://code.visualstudio.com/download

```
# Editor visual studio code
# Datei / einstellungen / User settings
{
  "workbench.iconTheme": "material-icon-theme",
  "powershell.powerShellExePath": "C:\\WINDOWS\\System32\\
      WindowsPowerShell\\v1.0\\powershell.exe",
  "editor.tabSize": 2,
  "php.executablePath": "C:/xampp/php/php.exe",
  "files.eol": "\n",
  "git.autofetch": true,
  "python.pythonPath": "D:\\anaconda\\python.exe",
  "window.zoomLevel": 1,
}
```

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

```
# Shell: Git version
git --version
```

Imagemagick: https://www.imagemagick.org/script/download.php#windows
Repository ordnerpaket-notizenDummy-v02 von Github downloaden

```
# Shell: Kopie downloaden
$ git clone https://github.com/ju-bw/ordnerpaket-notizenDummy-v02.git .
```

1.1.3 neues Repository auf github anlegen

```
# https://github.com/new
# github: Create a new repository
# Repository name = ordnerpaket-notizenDummy-v02
# Shell: Git Befehle
# ".gitconfig", ".gitignore" konfigurieren und erstellen
git init
git add .
git commit -am "Projekt start"
git remote add origin https://github.com/ju-bw/ordnerpaket-notizenDummy-
    v02.git
git push -u origin master
git status
git pull
git push
git status
```

1.1.4 Markdown Dokumente / Notizen

Markdown Dokumente / Notizen im Ordner "md/neu.md" erstellen.
Beachte das *min. zwei Markdowndateien* vorhanden sein müssen.
Powershellscript "docKonverter-v02.ps1" erstellt LaTeX - pdf und html files.

```
# Editor - Powershellscript "docKonverter-v02.ps1" anpassen
  ### Projekt
  # anpassen
  $thema = "ordnerpaket-notizenDummy-v02" # Thema
  $bildformat = "svg"    # Bildformate: svg, jpg, png
  $codeformat = "sh"     # Codeformate: c, cpp, sh, py, ps1
  $language = "Powershell"   # Latex-Code:  C, [LaTeX]TeX, Bash, Python,
      Powershell
# Shell: Script ausfuehren
$ ./docKonverter-v02.ps1
```

1.1.5 Bilder optimieren

JPG Bilder in den Ordner "imgOriginal/" kopieren.
Powershellscript "#imgWeb.ps1" optimiert Fotos für das Web und die PDF Datei.

```
# Shell: Script ausfuehren
$ ./imgWeb.ps1
```

2. Kapitel

2.1 Projekthilfe

% ju – https://bw1.eu – 10-Okt-18 – projekthilfe.md

2.1.1 Editor - Visual Studio Code

Tastenkombination und Einstellungen

```
Editor - Visual Studio Code
  - Shell öffnen: file Auswahl   <Alt+Strg+O>
  - mehrfaches Editieren         <Alt+Mausklick>
  - Einzug: 2 (Leerzeichen), Codierung: UTF-8, Zeilenende: LF (Linux)
```

2.1.2 Befehle Pandoc

Pandoc - universeller Dokumentenkonverter

```
$ # Shell oeffnen
# Pandoc: dokumentenkonverter
pandoc text.md -o text.pdf
pandoc -s text.md -c main-design.css -o text-mit-css.html
pandoc text.md -o text.html
pandoc text.md -o text.tex
# aufraeumen
rm *.log
rm *.out
rm *.aux
rm *.synctex.gz
```

2.1.3 Dokumentenkonverter

Latex

pandoc text.md -o text.tex

```
Editor "text.tex" oeffnen -> Suchen und Ersetzen
TeXworks "dummy.tex" oeffnen -> pdflatex
```

html

```
pandoc text.md -o text.html
```

HTML5 mit CSS

```
pandoc -s text.md -c design.css -o text-mit-css.html
```

pdf

```
pandoc text.md -o text.pdf
```

Word

```
pandoc text.md -o text.docx
```

3. Kapitel

3.1 Git Workflow

% ju - 18.10.18

3.1.1 git log

```
git log --oneline
git log --pretty=format:'%h ; %an ; %ad ; %s' --date=short
git log --graph
git log --oneline --graph --decorate HEAD master
```

3.1.2 branch erstellen - wechseln - löschen

```
git branch
git checkout -b entwicklung # erstellen u. wechseln
git checkout master         # wechsel auf master
git checkout entwicklung    # wechsel auf entwicklung
git branch -d entwicklung   # loeschen
```

3.1.3 Entwicklungs Zweig

Projekt neu erstellen

```
# git config u. .gitignore
# +++ files erstellen
git init
git status
git add . # lokale aenderungen
git commit -am"projekt start"
# Entwicklungszweig erstellen u. wechseln
git checkout -b entwicklung
git log --oneline
  b7a87de (HEAD -> entwicklung, master) projekt start
```

Projekt bearbeiten

```
git branch
  * entwicklung
    master
# auf "entwicklung" branch wechseln
git checkout entwicklung    # wechsel auf entwicklung
# +++ files bearbeiten
git status
git add . # lokale aenderungen
```

```
git commit -am"kommentar"
git branch
git checkout master          # wechsel auf master
git log --oneline
#git diff file
git merge entwicklung        # zusammenfuehren
git log --oneline
```

Version erstellen

```
# auf "master" branch wechseln
git tag v-01 letzter commit
```

Letzter Stand rückgängig machen

```
git log --oneline
git revert "letzterCommit"
git reset --hard "zielCommit"
```

3.1.4 Arbeiten im Team

3.1.4.1 lokaler Server

lokales Repository erstellen (zentrales Repository)

```
# auf "master" branch wechseln
git clone --bare . ../projekt.git
git status
# aenderungen veroeffentlichen
git pull ../projekt.git master
git push ../projekt.git master
git log --oneline
  f3d7d67 (HEAD -> master) Merge branch 'entwicklung'
```

Arbeitskopie erstellen

```
git clone ../projekt.git .
```

3.1.4.2 github Server

Remote Repository auf github erstellen
https://github.com/ju-bw?tab=repositories

```
# auf "master" branch wechseln
# aenderungen veroeffentlichen
git remote add origin https://github.com/ju-bw/projekt.git
git push -u origin master
git status
```

```
git pull
git push
git log --oneline
  f3d7d67 (HEAD -> master, origin/master) Merge branch 'entwicklung'
```

Arbeitskopie erstellen

```
git clone https://github.com/ju-bw/projekt.git .
```

4. Kapitel

4.1 Abbildungen

4.1.1 logo-black

(Abbildung 4.1 logo-black).

Abbildung 4.1: logo-black

4.1.2 logo

(Abbildung 4.2 logo).

4.1.3 titelbild-black

(Abbildung 4.3 titelbild-black).

4.1.4 titelbild

(Abbildung 4.4 titelbild).

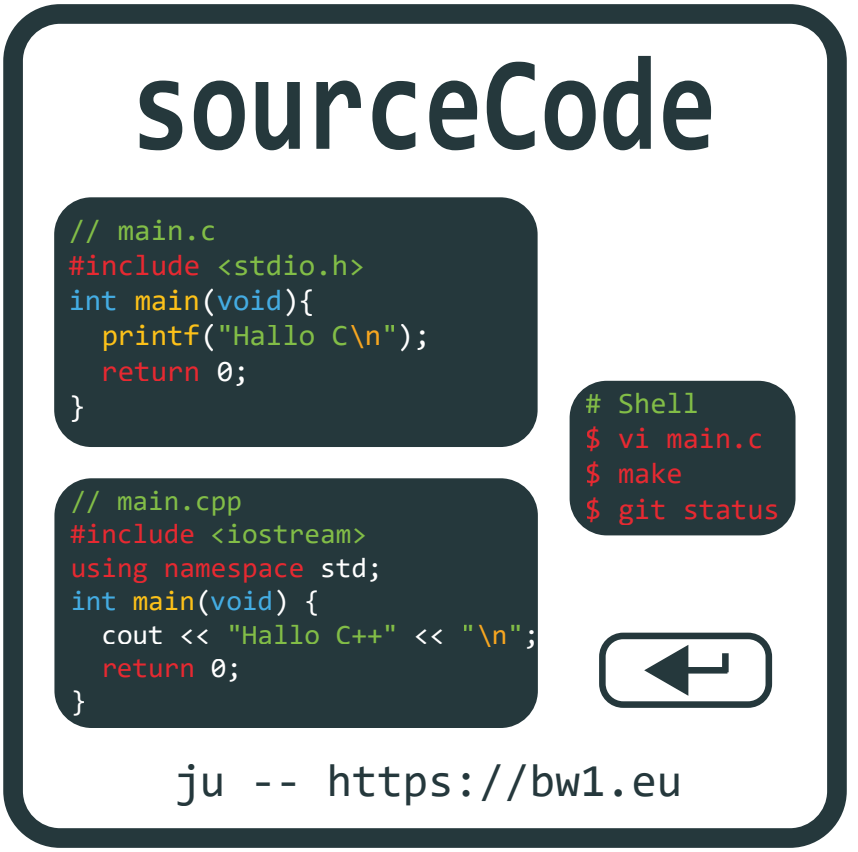

Abbildung 4.2: logo

Abbildung 4.3: titelbild-black

Abbildung 4.4: titelbild

5. Kapitel

5.1 Quellcode

5.1.1 docKonverter-v02

(Quelltext 5.1 docKonverter-v02).

```
# PowerShell Script:  ju -- https://bw1.eu -- 10-Okt-18  -- docKonverter-
    v02.ps1
# Shell: Script ausfuehren
# $ ./docKonverter-v02.ps1

<#
  akt. PowerShell: https://github.com/PowerShell/PowerShell/releases
  PS-Version: $PSVersionTable

  Markdown Dokumente / Notizen im Ordner "md/neu.md" erstellen.
  Beachte das *min. zwei Markdowndateien* vorhanden sein müssen.
  **Powershellscript** "docKonverter-v02.ps1" erstellt LaTeX - pdfs und
      html files.
#>

<#
  Editor - Visual Studio Code
    - Shell öffnen: file Auswahl   <Alt+Strg+O>
    - mehrfaches Editieren         <Alt+Mausklick>
    - Einzug: 2 (Leerzeichen), Codierung: UTF-8, Zeilenende: LF (Linux)
#>

<### Git
# repository auf github notwendig!
git init
git add .
git commit -am "Projekt start"
git remote add origin https://github.com/ju-bw/$thema.git
git push -u origin master
git status

git add .
git commit -a # vim Texteingabe: <i> / beenden mit <ESC : wq>
#git commit --amend # letzten Commit rueckgaengig machen
git pull
git push
#git log  # less beenden mit <Shift+q>
# --global: Datei /C/Users/jan/.gitconfig
# loa = log --graph --decorate --pretty=oneline --abbrev-commit --all
git loa
#>

Clear-Host # cls
```

```
### Projektverzeichnis
# anpassen
#$work = "C:/projekte/"
#cd $work

### Zeit
$timestampArchiv = Get-Date -Format 'yyyy-MMM-dd' # 2018-Okt-11
#$timestampArchiv = Get-Date -Format 'yMd'         # 181011
#$timestampFile = Get-Date -Format 'dd-MMM-yyyy'   # 11-Okt-2018
$timestampFile = Get-Date -Format 'd-MMM-y'       # 11-Okt-18

$autor = "ju -- https://bw1.eu -- $timestampFile"


### Projekt
# anpassen
$thema = "ordnerpaket-notizenDummy-v02" # Thema
$bildformat = "svg"    # Bildformate: svg, jpg, png
$codeformat = "sh"     # Codeformate: c, cpp, sh, py, ps1
$language = "Bash"   # Latex-Code:  C, [LaTeX]TeX, Bash, Python,
    Powershell
#
#$doc = "doc"
$tex = "tex"
$md = "md"
$html = "html"
$css = "css"
$wp = "wordpress"
$img = "img"
$code = "code"
$archiv = "archiv"
$pdf = "pdf"
#$docx = "docx" # Word
$imgOriginal = "imgOriginal"
$imgWebTex = "imgWebTex"
$content = "content"
# ext. dateien
$webDesign = "$css/design.css"
$texDummyArtikel = "$content/texDummyArtikel.tex"
$texDummyBook = "$content/texDummyBook.tex"
$texDummyPrint = "$content/texDummyPrint.tex"
$inhalt = "$content/inhalt.tex" # Inhalt book.tex & print.tex

### info
$info = "Auswahlmenue"
# Menue
$auswahl =  @'
  (1) artikel.pdf
  (2) book.pdf
  (3) print.pdf
  (4) alle Abbildungen.tex
  (5) alle Quellcodedateien.tex
```

```
  (6) backup - "../$archiv/$timestampArchiv-$thema.zip"
  (7) git - Repository auf github notwendig!
  (8) imgWeb.ps1 # ext. Script - Bilder optimieren (Latex/Web)
  (9) html
  (10) pandoc & suchen/ersetzen - Achtung: min. zwei Markdown Dateien
       notwendig !!!
  (11) Projekt reset
  (12) Projekt neu
  (13) Beenden
'@

# Menueeintraege
$auswahl_von = 1
$auswahl_bis = 13

### Fragen
$frage      = " n$ Bitte eine Auswahl treffen."
$frageTitel = " n$ Bitte einen Haupttitel eingeben."

### Fehler
$ErrorEingabe1 = " n$ Eingabefehler, Die richtige Zahl eingegeben!"
$ErrorEingabe2 = " n$ Eingabefehler, keine Zahl eingegeben!"


### Haupttitel fuer Latex
# Usereingabe
#[string]$thema = Read-Host 'Eingabe - [Haupttitel]'

<### Funktionen #>
# Verzeichnis erstellen, wenn nicht vorhanden
### Funktionsaufruf: ordnerErstellen
function ordnerErstellen{
  "+++ Verzeichnis erstellen, wenn nicht vorhanden"
  #if(!(test-path $docx)){md $docx} # Word
  if(!(test-path $html)){md $html/$wp}
  if(!(test-path $pdf)){md $pdf}
  if(!(test-path $tex)){md $tex}
  if(!(test-path $imgOriginal)){md $imgOriginal}
  if(!(test-path $code)){md $code}
  if(!(test-path $img)){md $img}
}

# Struktur
### Funktionsaufruf: trennLinie $max $muster
function trennLinie{
  param(
    [int]$max,
    [char]$muster
  )
  for($i=1; $i -le $max; $i++){
    $linie += $muster #array
  }
  Write-Host " n$linie"
```

```
}

# aufraeumen
### Funktionsaufruf: texAufraeumen
function texAufraeumen{
  ### loescht ordner, wenn vorhanden, recursiv, schreibgeschützt,
      versteckt (unix)
  if(test-path ./*.pdf){rm *.pdf -force}
  if(test-path ./*.log){rm *.log -force -recurse}
  if(test-path ./*.out){rm *.out -force -recurse}
  if(test-path ./*.aux){rm *.aux -force -recurse}
  if(test-path ./$tex/*.aux){rm ./$tex/*.aux -force -recurse}
  if(test-path ./$content/*.aux){rm ./$content/*.aux -force -recurse}
  if(test-path ./*.synctex*){rm *.synctex* -force -recurse}
  if(test-path ./*.bbl){rm *.bbl -force -recurse}
  if(test-path ./*.bcf){rm *.bcf -force -recurse}
  if(test-path ./*.blg){rm *.blg -force -recurse}
  if(test-path ./*.run*){rm *.run* -force -recurse}
  if(test-path ./*.toc){rm *.toc -force -recurse}
}

# pdflatex
# "*main.tex"
### Funktionsaufruf: texPdfLatex "*main.tex"
function texPdfLatex{
  param(
    [string]$filter
  )
  #$filter = "*main.tex"
  [array]$array = ls ./ $filter -Force
  for ($x=0; $x -lt $array.length; $x++){#-lt=kleiner
    $name = "$($array[$x])"              # file.tex
    $basename = "$($array.BaseName[$x])" # file
    # pdflatex "main.tex"
    pdflatex $name
    biber $basename
    pdflatex $name
    pdflatex $name
  }
}


# Latex - Artikel
### Funktionsaufruf: texArtikel
function texArtikel{
  $filter = "*.tex"
  [array]$array = ls ./$tex $filter -Recurse -Force
  $artikel = "$texDummyArtikel"
  $readFile = @(Get-content "$artikel" -Encoding UTF8)
  # array auslesen
  for($n=0; $n -lt $array.length; $n++){   # kleiner
    #$name = "$($array[$n])"              # file.tex
    $basename = "$($array.BaseName[$n])" # file
```

```
    #"$n - $name"
    # schreibe jeweils in datei
    $artikel = "$basename-main.tex"
    $readFile | Out-File "$artikel" -Encoding UTF8
    # Suchen und Ersetzen
    $suchen = "DUMMY" # regulaerer Ausdruck
    $ersetzen = "$basename"
    # regulaerer Ausdruck
    (Get-Content "$artikel") | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content "$artikel"
  }
}

# Inhalt fuer book.tex & print.tex
### Funktionsaufruf: texInhaltBookPrint
function texInhaltBookPrint{
  # schreibe in datei
  $text = "% Inhalt  n% $autor"
  $text | Set-Content $inhalt
  $filter = "*.tex"
  [array]$arrayTEX = ls $tex $filter -Recurse -Force
  # array auslesen
  for($n=0; $n -lt $arrayTEX.length; $n++){   # kleiner
    #$name = "$($arrayTEX[$n])"               # file.tex
    $basename = "$($arrayTEX.BaseName[$n])" # file
    #"$n - $basename"
    # schreibe in datei
    $text = "\chapter{Kapitel}  n\input{tex/$basename} n"
    $text | Add-Content $inhalt
  }
}

# html
### Funktionsaufruf: htmlFiles $fileTitel $fileHTML $fileTyp $filter
function htmlFiles{
  param(
    [string]$fileTitel,
    [string]$fileHTML,
    [string]$fileTyp,
    [string]$filter
  )
  $textHTML = "<!--+++ $autor +++-->
<!DOCTYPE html>
<html><head>
  <meta charset= "UTF-8 " />
  <title>$fileTitel</title><!-- Titel -->
  <meta name= "description " content= "$fileTitel " /><!-- Beschreibung
      -->
  <meta name= "viewport " content= "width=device-width, initial-scale
      =1.0, user-scalable=yes " />
  <link rel= "stylesheet " href= "../$webDesign " />
</head><body>
<!-- Inhalt -->
```

```
<h1>$fileTitel</h1>
<p>Inhalt</p>
<ul class= "nav "><!-- Liste -->"
  # schreibe in datei
  $textHTML | Set-Content ./$html/$fileHTML
  #$filter = "*.pdf"
  [array]$array = ls ./$fileTyp $filter -Force
  # array auslesen
  #$picnummer = 1
  for($n=0; $n -lt $array.length; $n++){   # kleiner
    $name = "$($array[$n])"              # file.tex
    #$basename = "$($array.BaseName[$n])" # file
    #"$n - $basename"
    $textHTML = "   <li><a href= "../$fileTyp/$name ">$name</a></li>"
    # schreibe in datei hinzu
    $textHTML | Add-Content ./$html/$fileHTML
    #$picnummer++
  }
  $textHTML = "</ul> n<!-- Ende Inhalt --> n</body></html>"
  # schreibe in datei hinzu
  $textHTML | Add-Content ./$html/$fileHTML
}
### Ende Funktionen

# Verzeichnis erstellen, wenn nicht vorhanden
### Funktionsaufruf:
ordnerErstellen

<### +++ while Schleife +++ ###>

[bool]$nochmal = $true
while($nochmal){
  ### Funktionsaufruf:
  trennLinie 33 *
  $autor
  Write-Host " n$info n============n"
  $auswahl      # auswahlmenü
  trennLinie 33 *

  # Fehlerbehandlung
  try {
      # Usereingabe
      [int]$varAuswahl = Read-Host 'Eingabe - [Zahl]'
      # Eingabefehler: menüeinträge von - bis
      if( $varAuswahl -lt $auswahl_von -or $varAuswahl -gt $auswahl_bis )
           {
          $ErrorEingabe1
      }
  }
  catch {
      # Eingabefehler: keine Zahl
      $ErrorEingabe2
      [int]$varAuswahl = 0 # initialisieren, sonst Fehler
```

```
}

# artikel.pdf
if($varAuswahl -eq 1){ # gleich
  # Latex - Artikel
  ### Funktionsaufruf:
  texArtikel

  # pdflatex
  ### Funktionsaufruf:
  texPdfLatex "*main.tex"

  # Kopie
  cp *.pdf $pdf/

  # aufraeumen
  ### Funktionsaufruf:
  texAufraeumen
}
# book.pdf
if($varAuswahl -eq 2){ # gleich
  # Latex - Artikel
  ### Funktionsaufruf:
  texArtikel

  ### PDF book
  # lese aus datei
  $readFile = @(Get-content "$texDummyBook" -Encoding UTF8)
  # schreibe in datei
  $book = "$thema-book"
  $readFile | Set-Content "$book.tex"
  # Suchen und Ersetzen
  $suchen = "Haupttitel" # regulaerer Ausdruck
  $ersetzen = "$thema"
  (Get-Content "$book.tex") | Foreach-Object {$_ -replace "$suchen", "
       $ersetzen"} | Set-Content "$book.tex"

  # Inhalt fuer book.tex & print.tex
  ### Funktionsaufruf:
  texInhaltBookPrint

  # pdflatex "main.tex"
  pdflatex "$book.tex"
  biber "$book"
  pdflatex "$book.tex"
  pdflatex "$book.tex"

  # Kopie
  cp *.pdf $pdf/

  # aufraeumen
  ### Funktionsaufruf:
  texAufraeumen
```

```
}

#  print.pdf
if($varAuswahl -eq 3){ # gleich
  # Latex - Artikel
  ### Funktionsaufruf:
  texArtikel

  ### PDF print
  # lese aus datei
  $readFile = @(Get-content "$texDummyPrint" -Encoding UTF8)
  # schreibe in datei
  $print = "$thema-print"
  $readFile | Set-Content "$print.tex"
  # Suchen und Ersetzen
  $suchen = "Haupttitel" # regulaerer Ausdruck
  $ersetzen = "$thema"
  (Get-Content "$print.tex") | Foreach-Object {$_ -replace "$suchen", "
       $ersetzen"} | Set-Content "$print.tex"

  # Inhalt fuer book.tex & print.tex
  ### Funktionsaufruf:
  texInhaltBookPrint

  # pdflatex "main.tex"
  pdflatex "$print.tex"
  biber "$print"
  pdflatex "$print.tex"
  pdflatex "$print.tex"

  # Kopie
  cp *.pdf $pdf/

  # aufraeumen
  ### Funktionsaufruf:
  texAufraeumen
}

# alle Abbildungen.tex
if($varAuswahl -eq 4){ # gleich
  ### alle Abbildungen
  # schreibe in datei
  $file =  "Abbildungen.tex"
  $text = "\section{Abbildungen}\label{abbildungen} n"
  $text | Set-Content $tex/$file
  $filter = "*.pdf"
  [array]$arrayBild = ls -Path $img -Filter $filter -Recurse -Force
  # array auslesen
  for($n=0; $n -lt $arrayBild.length; $n++){ # kleiner
    $name = "$($arrayBild[$n])"               # file.tex
    $basename = "$($arrayBild.BaseName[$n])" # file
    #"$n - $basename"
    # schreibe in datei
```

```
      $text = "\subsection{$basename}\label{} n
% Bild Referenz
(\autoref{fig:$basename} $basename). % bildname anpassen!
% Bild
\begin{figure}[!hb] % hier
  \centering
  \includegraphics[width=0.8\linewidth]{$img/$basename.pdf}
  % ====================
  \caption{$basename}   % Caption  anpassen!
  \label{fig:$basename} % Referenz anpassen!
  % ====================
\end{figure} n"
      $text | Add-Content $tex/$file
    }
  }

  # alle Quellcodedateien.tex
  if($varAuswahl -eq 5){ # gleich
    ### alle Quellcode dateien
    #$codeformat = "py"     # Codeformate: c, cpp, sh, py
    #$language = "Python"   # Latex-Code:  C, [LaTeX]TeX, Bash, Python
    # schreibe in datei
    $file =  "Quellcode.tex"
    $text = "\section{Quellcode}\label{quellcode} n"
    $text | Set-Content $tex/$file
    $filter = "*.$codeformat"
    [array]$arrayCode = ls -Path $code -Filter $filter -Recurse -Force
    # array auslesen
    for($n=0; $n -lt $arrayCode.length; $n++){ # kleiner
      $name = "$($arrayCode[$n])"              # file.tex
      $basename = "$($arrayCode.BaseName[$n])" # file
      #"$n - $basename"
      # schreibe in datei
      $text = "\subsection{$basename}\label{} n
% Quellcode Referenz
(\autoref{code:$basename} $basename). % codename anpassen!
% Quellcode aus ext. Datei
\lstset{language=$language}% cpp, [LaTeX]TeX, Bash, Python
\lstinputlisting[%numbers=left, frame=l, framerule=0.1pt,
  % ====================
  caption={Quellcode in $language, $basename}, % Caption  anpassen!
  label={code:$basename}                       % Referenz anpassen!
  % ====================
]{$code/$basename.$codeformat}% ext. Datei

\newpage n"
      $text | Add-Content $tex/$file
    }
  }

  # backup
  if($varAuswahl -eq 6){ # gleich
    ### Backup
```

```
  #robocopy $quelle $ziel /mir /e /NFL /NDL /NJH /TEE
  if(test-path "../$archiv/$timestampArchiv-$thema.zip"){
    rm "../$archiv/$timestampArchiv-$thema.zip" -force -recurse
  }
  ### Komprimieren
  ls ./ | Compress-Archive -Update -dest "../$archiv/$timestampArchiv-
      $thema.zip"
}

# git
if($varAuswahl -eq 7){ # gleich
  # git
  git status
  # Usereingabe
  "+++ Gibt es Aenderungen? Wenn ja, Repository auf github notwendig!
      siehe Readme.md"

  [char]$varGit = Read-Host 'Eingabe - [j/n]'
  if($varGit -eq  "n"){# gleich
    "keine Aenderung"
  }
  else{
    # repository auf github notwendig!
    git add .
    git commit -a # vim Texteingabe: <i> / beenden mit <ESC : wq>
    git pull
    git push
    git log --oneline  # less beenden mit <Shift+q>
    # Datei /C/Users/jan/.gitconfig
    # loa = log --graph --decorate --pretty=oneline --abbrev-commit --
        all
    #git loa
  }
}

# #imgWeb.ps1 # ext. Script
if($varAuswahl -eq 8){ # gleich
  ./imgWeb.ps1 # ext. Script
}

# html
if($varAuswahl -eq 9){ # gleich
  # html: alle-PDF-files.html
  "+++ alle-PDF-files.html"
  $fileTitel = "PDFs"
  $fileHTML  = "alle-PDF-files.html"
  $fileTyp   = $pdf
  $filter    = "*.$pdf"       # pdf
  ### Funktionsaufruf:
  htmlFiles $fileTitel $fileHTML $fileTyp $filter

  # html: alle-Abb-files.html
  "+++ alle-Abb-files.html"
```

```
    $fileTitel = "Abbildungen"
    $fileHTML  = "alle-Abb-files.html"
    $fileTyp   = $img
    $filter    = "*.$bildformat" # jpg, svg
    ### Funktionsaufruf:
    htmlFiles $fileTitel $fileHTML $fileTyp $filter

    # html: alle-Code-files.html
    "+++ alle-Code-files.html"
    $fileTitel = "Code"
    $fileHTML  = "alle-Code-files.html"
    $fileTyp   = $code
    $filter    = "*.$codeformat" # Codeformate: c, cpp, sh, py
    ### Funktionsaufruf:
    htmlFiles $fileTitel $fileHTML $fileTyp $filter



    "+++ alle-Pics.html"
    $titel = "Pics"
    $fileHTML =  "alle-Pics.html"
    $textHTML = "<!--+++ $autor +++-->
<!DOCTYPE html>
<html><head>
  <meta charset= "UTF-8 " />
  <title>$titel</title><!-- Titel -->
  <meta name= "description " content= "$titel " /><!-- Beschreibung -->
  <meta name= "viewport " content= "width=device-width, initial-scale
      =1.0, user-scalable=yes " />
  <link rel= "stylesheet " href= "../$webDesign " />
</head><body>
<!-- Inhalt -->
<h1>$titel</h1>
<p>Inhalt</p>"
    # schreibe in datei
    $textHTML | Set-Content ./$html/$fileHTML
    $filter = "*.$bildformat" # Bildformate: svg, jpg, png
    [array]$arrayAbb = ls ./$img $filter -Force
    # array auslesen
    $picnummer = 1
    for($n=0; $n -lt $arrayAbb.length; $n++){   # kleiner
      $name = "$($arrayAbb[$n])"              # file.tex
      $basename = "$($arrayAbb.BaseName[$n])" # file
      #"$n - $basename"
      $textHTML = "<!-- Abb. $picnummer -->
<a href= "../$img/$name ">
  <figure>
    <img class=scaled src= "../img/$name " />
    <figcaption>Abb. $picnummer : $name</figcaption>
  </figure>
</a>"
      # schreibe in datei hinzu
      $textHTML | Add-Content ./$html/$fileHTML
```

```
      $picnummer++
    }
    $textHTML = "<!-- Ende Inhalt --> n</body></html>"
    # schreibe in datei hinzu
    $textHTML | Add-Content ./$html/$fileHTML


    "+++ index.html - alle html-Seiten"
    $titel = "$thema"
    $fileHTML =  "index.html"
    $textHTML = "<!--+++ $autor +++-->
<!DOCTYPE html>
<html><head>
  <meta charset= "UTF-8 " />
  <title>$titel</title><!-- Titel -->
  <meta name= "description " content= "$titel " /><!-- Beschreibung -->
  <meta name= "viewport " content= "width=device-width, initial-scale
      =1.0, user-scalable=yes " />
  <link rel= "stylesheet " href= "./$webDesign " />
</head><body>
<!-- Inhalt -->
<h1>$titel</h1>
<p>Inhalt</p>
<ul class= "nav "><!-- Liste -->"
    # schreibe in datei
    $textHTML | Set-Content $fileHTML
    $filter = "*.html"
    [array]$arrayHTML = ls ./$html $filter -Force
    # array auslesen
    for($n=0; $n -lt $arrayHTML.length; $n++){   # kleiner
      $name = "$($arrayHTML[$n])"              # file.tex
      $basename = "$($arrayHTML.BaseName[$n])" # file
      #"$n - $basename"
      $textHTML = "   <li><a href= "./$html/$basename.html ">$basename.
          html</a></li>"
      # schreibe in datei hinzu
      $textHTML | Add-Content $fileHTML
    }
    $textHTML = "</ul> n<!-- Ende Inhalt --> n</body></html>"
    # schreibe in datei hinzu
    $textHTML | Add-Content $fileHTML
  }

  # Pandoc
  if($varAuswahl -eq 10){ # gleich
    # loeschen
    if(test-path ./Abbildungen-main.tex){
      rm ./Abbildungen-main.tex -force -recurse
    }
    if(test-path ./$tex/Abbildungen.tex){
      rm ./$tex/Abbildungen.tex -force -recurse
    }
    if(test-path ./Quellcode-main.tex){
```

```
  rm ./Quellcode-main.tex -force -recurse
}
if(test-path ./$tex/Quellcode.tex){
  rm ./$tex/Quellcode.tex -force -recurse
}

### Files umbenennen
cd $img
"+++ Files umbenennen"
# Dateiendung
ls -r | ren -NewName {$_.fullname -replace ".JPG", ".jpg"} -ea
     SilentlyContinue
ls -r | ren -NewName {$_.fullname -replace ".jpeg", ".jpg"} -ea
     SilentlyContinue
# Leerzeichen
ls -r | ren -NewName {$_.name -replace "_", "-"} -ea SilentlyContinue
ls -r | ren -NewName {$_.name -replace " ", ""} -ea SilentlyContinue
# webserver
#ls -r | ren -NewName {$_.name -replace "-", "_"} -ea
     SilentlyContinue
# Umlaute ergaenzen
cd ..


### Markdown in *.tex, *.pdf, *.html
$filter = "*.md"
[array]$array = ls -Path ./$md -Filter $filter -Recurse -Force
for ($x=0; $x -lt $array.length; $x++) { #-lt=kleiner
  $name = "$($array[$x])"              # file.md
  $basename = "$($array.BaseName[$x])" # file
  # pandoc text.md -o text.tex
  pandoc "./$md/$name" -o "./$tex/$basename.tex"
  #pandoc "./$md/$name" -o "./$pdf/$basename.pdf"
  # Word
  #pandoc "./$md/$name" -o "./$docx/$basename.docx"
  pandoc "./$md/$name" -o "./$html/$wp/$basename-$wp.html"
  pandoc -s "./$md/$name" -c "../$webDesign" -o "./$html/$basename.
       html"
}



### suchen und ersetzen in latex
$filter = "*.tex"
[array]$array = ls -Path ./$tex -Filter $filter -Recurse -Force
# array auslesen
for($n=0; $n -lt $array.length; $n++){   # kleiner
  $name = "$($array[$n])"              # file.tex
  $basename = "$($array.BaseName[$n])" # file
  #"$n - $name"
# \tightlist
$suchen = "\\tightlist" # regulaerer Ausdruck
$ersetzen = ""
```

```
    # regulaerer Ausdruck
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # \hypertarget
    $suchen = "\\hypertarget{" # regulaerer Ausdruck
    $ersetzen = "%% Ueberschrift: "
    # regulaerer Ausdruck
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # }{%
    $suchen = "}{%" # regulaerer Ausdruck
    $ersetzen = ""
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # }}
    $suchen = "}}" # regulaerer Ausdruck
    $ersetzen = "}"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # Bilder
    $suchen = "includegraphics" # regulaerer Ausdruck
    $ersetzen = "includegraphics[width=0.8\textwidth]"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # \begin{figure}[!hb] % hier
    $suchen = "\\begin{figure}" # regulaerer Ausdruck
    $ersetzen = "\begin{figure}[!hb] % hier"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # Quellcode
    $suchen = "\\begin\{verbatim\}" # regulaerer Ausdruck
    $ersetzen = "% Quellcode
\lstset{language=$language} % C, [LaTeX]TeX, Bash, Python
\begin{lstlisting}[%numbers=left, frame=l, framerule=0.1pt,%
% ======================
  caption={},            % Caption anpassen!
  label={code:codename}  % Label anpassen!
]% ======================
"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name

    $suchen = "\\end\{verbatim\}" # regulaerer Ausdruck
    $ersetzen = "\end{lstlisting}"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # Tabelle
    $suchen = "\[\]\{\@\{\}" # regulaerer Ausdruck
    $ersetzen = ""
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    #
    $suchen = "\@\{\}" # regulaerer Ausdruck
```

```
    # regulaerer Ausdruck
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    #
    $suchen = "siehe tab. " # regulaerer Ausdruck
    $ersetzen = "% Tabellenverweis
    (\autoref{tab:tabname}). % tabname = "
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    #
    $suchen = "\\begin\{longtable\}" # regulaerer Ausdruck
    $ersetzen = "% Tabelle
\begin{table}[!ht] % hier
\centering
\rowcolors{1}{}{lightgray!20} % Farbe
%\begin{tabularx}{\textwidth}{XXX}   % auto. Spaltenumbruch
%\begin{tabular}{p{4cm}p{4cm}p{4cm}} % Spaltelaenge fest, auto.
     Spaltenumbruch, Text nur linksbuendig
\begin{tabular}{"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    #
    $suchen = "\\end\{longtable\}" # regulaerer Ausdruck
    $ersetzen = "%\end{tabularx}
\end{tabular}
% ========================
\caption{}                  % Caption  anpassen!
\label{tab:tabname}         % Referenz anpassen!
% ========================
\end{table}"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    #
    $suchen = "\\toprule" # regulaerer Ausdruck
    $ersetzen = "}%
\toprule
%\rowcolor{orange!90} % Farbe"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    #
    $suchen = "\\endhead" # regulaerer Ausdruck
    $ersetzen = ""
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    # Ersetze: \tabularnewline
    $suchen = "\\tabularnewline" # regulaerer Ausdruck
    $ersetzen = " \\"
    (Get-Content ./$tex/$name) | Foreach-Object {$_ -replace "$suchen", "
        $ersetzen"} | Set-Content ./$tex/$name
    }


    ### suchen u. ersetzen in html
```

```
  $filter = "*.html"
  [array]$array = ls ./$html $filter -Force
  # array auslesen
  for($n=0; $n -lt $array.length; $n++){   # kleiner
    $name = "$($array[$n])"              # file.tex
    $basename = "$($array.BaseName[$n])" # file
    #"$n - $name"
  # Bilder pdf in jpg
  $suchen = "pdf " /><fig" # regulaerer Ausdruck
  $ersetzen = "$bildformat " width= "400 "/><fig"
  # regulaerer Ausdruck
  (Get-Content ./$html/$name) | Foreach-Object {$_ -replace "$suchen",
       "$ersetzen"} | Set-Content ./$html/$name
  # Bild pfad
  $suchen = "src= "img" # regulaerer Ausdruck
  $ersetzen = "src= "../img"
  # regulaerer Ausdruck
  (Get-Content ./$html/$name) | Foreach-Object {$_ -replace "$suchen",
       "$ersetzen"} | Set-Content ./$html/$name
  # Bild skalieren: <embed src=   =>  <img class=scaled src=
  $suchen = "<embed src=" # regulaerer Ausdruck
  $ersetzen = "<img class=scaled src="
  (Get-Content ./$html/$name) | Foreach-Object {$_ -replace "$suchen",
       "$ersetzen"} | Set-Content ./$html/$name
  }

}

# projekt reset
if($varAuswahl -eq 11){ # gleich
  ### loescht ordner, wenn vorhanden, recursiv, schreibgeschützt,
       versteckt (unix)
  if(test-path $html){rm $html -force -recurse}
  if(test-path $pdf){rm $pdf -force -recurse}
  if(test-path $tex){rm $tex -force -recurse}
  if(test-path $imgWebTex){rm $imgWebTex -force -recurse}
  if(test-path $imgOriginal){rm $imgOriginal -force -recurse}
  # git
  <#if(test-path .git){
    rm .git -force -recurse
    git init
    git add .
    git commit -am "Projekt start"
    git remote add origin https://github.com/ju-bw/$thema.git
    git push -u origin master
    git status
    git log --oneline
  }
  #>
  # aufraeumen
  ### Funktionsaufruf:
  texAufraeumen
  if(test-path ./*.tex){rm *.tex -force}
```

```
    if(test-path ./*.html){rm *.html -force}
  }

  # projekt neu
  if($varAuswahl -eq 12){ # gleich
    # Verzeichnis erstellen, wenn nicht vorhanden
    ### Funktionsaufruf:
    ordnerErstellen
  }

  # Beenden
  if($varAuswahl -eq 13){ # gleich
      $nochmal = 0;# schleife beenden
  }

}

### Win - Explorer öffnen
#& explorer

### Shell offen halten
Write-Host $openShell
Read-Host "Beenden mit Enter ..."
```

Quelltext 5.1: Quellcode in Bash, docKonverter-v02

5.1.2 imgWeb

(Quelltext 5.2 imgWeb).

```
# PowerShell Script:  ju -- https://bw1.eu -- 10-Okt-18  -- imgWeb.ps1
# Shell: Script ausfuehren
# $ ./#imgWeb.ps1

<#
  **JPG Bilder** in den Ordner "imgOriginal/" kopieren.
  **Powershellscript** "#imgWeb.ps1" optimiert Fotos für das Web und die
       PDF Datei.
#>

<#
  Editor - Visual Studio Code
    - Shell öffnen: file Auswahl   <Alt+Strg+O>
    - mehrfaches Editieren         <Alt+Mausklick>
    - Einzug: 2 (Leerzeichen), Codierung: UTF-8, Zeilenende: LF (Linux)
#>

Clear-Host # cls

### Projektverzeichnis
# anpassen
#$work = "C:/projekte/"
#cd $work

### Zeit
#$timestampArchiv = Get-Date -Format 'yyyy-MMM-dd' # 2018-Okt-11
#$timestampArchiv = Get-Date -Format 'yMd'          # 181011
#$timestampFile = Get-Date -Format 'dd-MMM-yyyy'   # 11-Okt-2018
$timestampFile = Get-Date -Format 'd-MMM-y'       # 11-Okt-18
#
$autor = "ju -- https://bw1.eu -- $timestampFile"
$autor

#$aufloesung = '512x512' # anpassen!!!!!!!!
$aufloesungTex = '728x516'  # 1224x792 B5 = 728x516
$aufloesungWeb = '1920x1080' # 1920x1080 1280x1024
$qualitaetWeb = '75%' # ImageMagik: 82% = Photoshop: 60%
$imgOriginal = 'imgOriginal'
$imgWebTex = 'imgWebTex'
$tmp = 'tmp'


# Usereingabe
"+++ Sind Bilder im Ordner '$imgOriginal' ?"
$var = Read-Host 'Eingabe - [j/n]'
if($var -eq  "n"){# gleich
  "Script wird beendet"
  exit
}
else{
```

```
" n+++ Verzeichnis erstellen oder loeschen"
# loescht ordner, wenn vorhanden, recursiv, schreibgeschützt,
     versteckt (unix)
if (test-path $tmp) { remove-item $tmp -force -recurse}
if (test-path $imgWebTex) { remove-item $imgWebTex -force -recurse}

# ordner erstellen
md $imgWebTex/$tmp
md $imgWebTex/$aufloesungTex

" n+++ Kopie erstellen: $imgOriginal => $imgWebTex/$tmp"
cp -Recurse -Force $imgOriginal/* $imgWebTex/$tmp

cd $imgWebTex

"+++ files umbennen"
cd $tmp
# Dateiendung
ls -r | ren -NewName {$_.fullname -replace ".JPG", ".jpg"} -ea
     SilentlyContinue
ls -r | ren -NewName {$_.fullname -replace ".jpeg", ".jpg"} -ea
     SilentlyContinue
# Leerzeichen
ls -r | ren -NewName {$_.name -replace "_", "-"} -ea SilentlyContinue
ls -r | ren -NewName {$_.name -replace " ", ""} -ea SilentlyContinue
# webserver
#ls -r | ren -NewName {$_.name -replace "-", "_"} -ea
     SilentlyContinue
# Umlaute
# ergaenzen
cd ..

# Funktionen
# Funktion: Bildqualitaet optimieren
# Funktionsaufruf: imgOpti $out/ $qualitaet $in/*
function imgOpti() {
    param (
        [string] $out,
        [string] $aufloesung,
        [string] $qualitaet,
        [string] $in
    )
    #mogrify -path $out/ -resize 300 $in/*
    # Einstellungen mit Optimierung
    # automatisch drehen: -auto-orient
    mogrify -path $out/ -filter Triangle -define filter:support=2 -
         auto-orient  -thumbnail  $aufloesung  -unsharp 0.25x0
         .25+8+0.065  -dither None -posterize 136 -quality $qualitaet
          -define jpeg:fancy-upsampling=off -define png:compression-
         filter=5 -define png:compression-level=9 -define png:
         compression-strategy=1 -define png:exclude-chunk=all -
         interlace none -colorspace sRGB  -strip $in/*
}
```

```
    # Funktion: Bildaufloesung aendern
    # Funktionsaufruf: imgResize $out/ $aufloesung $in/*
    function imgResize() {
        param (
            [string] $out,
            [string] $aufloesung,
            [string] $in
        )
        #mogrify -path $out/ -resize 300 $in/*
        # Einstellungen mit Optimierung
        mogrify -path $out/ -thumbnail $aufloesung $in/*
    }

    "+++ pics umbenennen - Ordner-001.jpg"
    exiftool -P -fileOrder datetimeimgOriginal '-fileName<${directory
        }%-.3nc.%le' $tmp/* -r

    ### Web und Latex
    # Bildqualitaet optimieren
    # Funktionsaufruf: imgOpti $out/ $qualitaet $in/*
    "+++ Web - Bildaufloesung $aufloesungWeb Bildqualitaet  $qualitaetWeb
        "
    imgOpti ./ $aufloesungWeb $qualitaetWeb $tmp

    cd ..

    # Bildaufloesung aendern
    # Funktionsaufruf: imgResize $out/ $aufloesung $in/*
    "+++ LaTeX - Bildaufloesung $aufloesungTex"
    #imgResize "$imgWebTex/$aufloesung" $aufloesung $imgWebTex
    imgResize ./$imgWebTex/$aufloesungTex $aufloesungTex $imgWebTex

    cd $imgWebTex/

    "+++ LaTeX - Bilder in pdf umwandeln"
    mogrify -format pdf $aufloesungTex/*.jpg
    cp $aufloesungTex/*.pdf ./

    # Komprimiert den Inhalt eines Verzeichnisses
    #ls $ordner | Compress-Archive -Update -dest "$archiv/$ordner.zip"

    # Kopie loeschen
    rm $tmp -force -recurse
    rm $aufloesungTex -force -recurse

    cd ..
}
```

Quelltext 5.2: Quellcode in Bash, imgWeb