Zur Nutzung von `makefile`-Dateien

Lukas C. Bossert

Größere LaTeX-Projekte mit vielen Dateien zu managen ist nicht immer einfach oder man muss verschiedene Schritte stets manuell ausführen. Beispielsweise wenn man die PDF zusätzlich noch in einer komprimierten Fassung haben möchte oder wissen muss, auf welchen Seiten Farbinformationen im PDF hinterlegt sind.

Die folgenden Ausführungen beziehen sich auf GNUmake (Unix/macOS). Für Windows gibt es eine entsprechende Variante |nmake|.

Mittels einer `makefile`–Datei können mehrere Befehle gleichzeitig und/oder hintereinander ausgeführt werden, sodass verschiedene händische Arbeitsschritte abgenommen werden können.

Zunächst erläutere ich die Eigenschaften und den Aufbau einer `makefile`–Datei. Anschließend zeige ich an kleinen Beispielen, worin das Potenzial dieser unscheinbaren Datei liegt. Mir ist weniger daran gelegen, die (durchaus komplexe) Logik bei den Abhängigkeiten von »Ziel« und »Quelle« (s. u.) zu durchdringen, als vielmehr einen praktisch orientierten Einblick zu geben.

Der Aufbau einer minimalen `makefile`–Datei

Eine `makefile`–Datei ist eine schlichte Textdatei *ohne* Endung. Sie liegt idealerweise im gleichen Ordner wie die Hauptdatei des LaTeX-Projekts. Sie kann mit Variablen arbeiten und man kann alle Befehle ausführen lassen, die man auch im Terminal eingeben kann. Dies sind die zwei wichtigsten Merkmale, die wir gleich nutzen werden.

Zunächst definieren wir die Variable |PROJECT|, die den Dateinamen der Hauptdatei unseres LaTeX-Projekts beinhaltet.

```
1 PROJECT  = dtk-make-bossert
```

Nun wollen wir den Arbeitsschritt zum Erstellen der PDF einbauen.

```
1 all:
2     lualatex $(PROJECT)
```

Mit |all:| wird ein »Ziel« angegeben. In diesem Fall ist es die Standardausführung, wenn keine weiteren Angaben beim Ausführen der `makefile`–Datei gemacht werden. Alle folgenden Zeilen, die zu diesem »Ziel« gehören, werden mit einem Tab eingerückt. Mit |lualatex $(PROJECT)| wird die oben definierte Variable aufgerufen, sodass |lualatex dtk-make-bossert| ausgeführt wird.

Um die `makefile`–Datei auszuführen, navigiert man im Terminal zum Hauptordner des LaTeX-Projects und führt lediglich den Befehl |make| aus.

Weitere Variablen und Arbeitsanweisen in der `makefile`–Datei

Nach dieser kurzen Einführung können wir verschiedene »Ziele« basteln, um sie bei Bedarf oder immer ausführen zu lassen.

Es empfielt sich anzugeben, wo |make| die Shell findet. Dies erfolgt mit einer Variable.

```
SHELL = bash
```

Anschließend führen wir noch ein paar Farben ein, um die Lesbarkeit der Informationsdarstellung zu erhöhen.

```
# Colors
RED   = \033[0;31m
CYAN  = \033[0;36m
NC    = \033[0m # No color
echoPROJECT = @echo -e "$(CYAN)**** $(PROJECT) ****$(NC) \n"
```

Die letzte Variable gibt im Terminal den Projectnamen farblich aus.

Als erstes »Ziel« definieren wir die Erstellung des Artikels, wofür wir eine weitere Variable nutzen, die das aktuelle Datum abruft.

```
DATE  = $(shell /bin/date "+%Y-%m-%d")
```

Jetzt das »Ziel« selbst.

```
article:
  $(echoPROJECT) "$(RED) * making article * $(NC)"
  latexmk -lualatex -quiet -f -cd -view=pdf -output-directory=tmp/$(DATE) $(
 ↪PROJECT).tex
  @cp tmp/$(DATE)/$(PROJECT).pdf .
  $(echoPROJECT) "$(RED) * article compiled * $(NC)"
```

Als erstes soll im Terminal angezeigt werden, welches »Ziel« von |make| gerade ausgeführt wird (Z.2) bzw. abgeschlossen wurde (Z.6). Anschließend wird das PDF mittels |latexmk| erstellt, wozu weitere Optionen angegeben sind: Um den Hauptordner von allen temporären Dateien frei zu halten, werden diese in ein separates, datumsspezifisches Verzeichnis erstellt. Damit können Zwischenstände schnell und leicht (v. a. im jeweiligen PDF) nachgeschaut werden. Die PDF wird schließlich in den Hauptordner kopiert (Z. 5). Mit dem Präfix |@| wird die auszuführende Befehlszeile nicht im Terminal angezeigt, lediglich deren Result. Mit |make article| lässt sich diese Passage direkt ansteuern und ausführen.

Besonders bei bildlastigen PDFs ist deren Dateigröße manchmal auch zu groß, um sie für Korrekturen etc. zu verschicken. Das PDF muss dann in einem weiteren Schritt komprimiert werden. Dieser Vorgang lässt sich ebenfalls von |make| mittels Ghostscript ausführen.[1]

```
minimize: article
  $(echoPROJECT) "$(RED) * minimizing article * $(NC)"
  @-mkdir archive
  @rm -f archive/$(PROJECT)-$(DATE)*.pdf
  gs \
  -sDEVICE=pdfwrite \
  -dCompatibilityLevel=1.4 \
  -dPDFSETTINGS=/printer \
  -dNOPAUSE \
  -dQUIET \
  -dBATCH \
  -sOutputFile=archive/$(PROJECT)-$(VERS).pdf \
  $(PROJECT).pdf
  $(echoPROJECT) "$(RED) * article minimized * $(NC)"
```

Zunächst wird ein Ordner |archive| erstellt (falls er schon existiert wird mit dem Präfix |-| zwar eine Fehlermeldung ausgegeben, diese führt jedoch nicht zum Abbruch des Befehls). Das PDF wird komprimiert und mit Datumsangabe im Ordner |archive| abgelegt.

```
count.colorpages: article
  $(echoPROJECT) "$(RED) * counting colored pages * $(NC)"
  @echo -e "Pages with color:"
  @ gs -o - -sDEVICE=inkcov $(PROJECT).pdf | grep -v "^ 0.00000  0.00000
 ↪0.00000" | grep "^ " | wc -l
  @ gs \
  -o \
  - \
  -sDEVICE=inkcov \
  $(PROJECT).pdf \
  |tail -n +5 \
  |sed '/^Page*/N;s/\n//'\
  |sed -E '/Page [0-9]+ 0.00000 0.00000 0.00000 / d' \
  | tee  $(PROJECT).csv
  $(echoPROJECT) "$(RED) * colored pages counted * $(NC)"
```

[1] Zu den einzelnen Optionen des Ghostscriptbefehls s. XXX

Der Einsatz von `make`