

Zur Nutzung von makefile-Dateien

Lukas C. Bossert

Größere L^AT_EX-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 L^AT_EX-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 L^AT_EX-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 L^AT_EX-Projcts 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 empfiehlt sich anzugeben, wo *make* die Shell findet. Dies erfolgt mit einer Variable.

```
1 SHELL = bash
```

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

```
1 # Colors
2 RED   = \033[0;31m
3 CYAN  = \033[0;36m
4 NC    = \033[0m # No color
5 echoPROJECT = @echo -e "$(CYAN) <$(PROJECT)>"
```

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.

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

Jetzt das »Ziel« selbst.

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

Das 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

Das »Ziel« ist *minimize* und als »Quelle« geben wir das oben formulierte »Ziel« *article* an. Das heißt, dass beim Aufruf von *minimize* zuerst das »Ziel« *article* ausgeführt wird – Dank *latexmk* wird nur bei veränderter *.tex*-Datei neu übersetzt. Somit wird gewährleistet, dass immer die neueste PDF-Version minimiert wird.

```

1 minimize: article
2   $(echoPROJECT) "$(RED) * minimizing article * $(NC)"
3   @-mkdir archive
4   @rm -f archive/${PROJECT}-${DATE}*.pdf
5   gs \
6   -sDEVICE=pdfwrite \
7   -dCompatibilityLevel=1.4 \
8   -dPDFSETTINGS=/printer \
9   -dNOPAUSE \
10  -dQUIET \
11  -dBATCH \
12  -sOutputFile=archive/${PROJECT}-${VERS}.pdf \
13  ${PROJECT}.pdf
14  $(echoPROJECT) "$(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.

Um auch zugleich den Status quo des L^AT_EX-Projekts festzuhalten, kann man alle notwendige Dateien tagesaktuell zippen. Somit hat man immer den letzten Tagesstand im Ordner *archive* gesichert. Dafür bedarf es noch ein paar Variablen, die wir vorweg definieren.

```

1 # zip
2 PWD  = $(shell pwd)
3 TEMP := $(shell mktemp -d -t tmp.XXXXXXXXXX)
4 TDIR = $(TEMP)/$(PROJECT)
5 VERS = $(shell /bin/date "+%Y-%m-%d---%H-%M-%S")
6 DATE = $(shell /bin/date "+%Y-%m-%d")

```

Das »Ziel« heißt *zip* und es wird wiederum zuerst *article* ausgeführt, um die aktuelle Projektversion zu zippen.

¹ Zu den einzelnen Optionen des Ghostscriptbefehls s. XXX

```

1 zip: article
2   $(echoPROJECT) "$(RED) * start zipping files * $(NC)"
3   @mkdir archive
4   @rm -f archive/$(PROJECT)-$(DATE).zip
5   @mkdir $(TDIR)
6   @cp $(PROJECT).{bib,tex,pdf} README.md makefile $(TDIR)
7   cd $(TEMP); \
8   zip -Drq $(PWD)/archive/$(PROJECT)-$(VERS).zip $(PROJECT)
9   $(echoPROJECT) "$(RED) * files zipped * $(NC)"

```

Möchte man sein PDF an eine Druckerei geben, braucht man die genaue Anzahl der Farbseiten plus der Auflistung der Farben. Es wäre fatal (und unnötig), dies bei größeren PDFs von Hand zu tun. Folgende Code gibt eine Tabulator getrennte csv-Datei mit der prozentualen Farbabdeckung von jeder Seite.² Damit kann man sehr leicht erkennen, ob das CMYK-Farbmodell korrekt ist und auf welchen Seiten **Cyan**, **Magenta** oder **Gelb** (CMY) verwendet wird.³

```

1 count.colorpages: article
2   $(echoPROJECT) "$(RED) * counting colored pages * $(NC)"
3   @ gs \
4   -o - \
5   -sDEVICE=inkcov \
6   $(PROJECT).pdf \
7   |tail -n +5 \
8   |sed '/^Page*/N;s/\n//'\
9   |sed -E '/Page [0-9]+ 0.00000 0.00000 0.00000 / d' \
10  | tee $(PROJECT).csv
11  @echo -e "Total amount of pages with color: "
12  @ gs -o - -sDEVICE=inkcov $(PROJECT).pdf | \
13  grep -v "^ 0.00000 0.00000 0.00000" | grep "^ " | wc -l
14  $(echoPROJECT) "$(RED) * colored pages counted * $(NC)"

```

Die csv-Datei mit der Liste der Farbseiten für diesen Artikel sieht dann so aus:

```

Page 1 0.00000 0.00000 0.00000 0.12856 CMYK OK
Page 2 0.00000 0.00000 0.00000 0.27076 CMYK OK
Page 3 0.00000 0.00000 0.00000 0.34784 CMYK OK
Page 4 0.00016 0.00035 0.00017 0.44522 CMYK OK
Page 5 0.00000 0.00000 0.00000 0.60695 CMYK OK
Page 6 0.00000 0.00000 0.00000 0.49144 CMYK OK
Page 7 0.00000 0.00000 0.00000 0.00608 CMYK OK

```

² <https://stackoverflow.com/a/28369599>

³ Man könnte diesen Code noch verbessern, indem man direkt eine kommaseparierte Liste ausgegeben bekommt, die die Seitenzahlen der Farbseiten aufzählt.

Damit haben wir nun ein paar hilfreiche »Ziele« und Vorgehensweisen kennengelernt, die wir nun in eine *makefile*-Datei schreiben werden.

```

1 PROJECT = dtk-make-bossert
2 SHELL = bash
3 MAKE = make
4 # zip
5 PWD = $(shell pwd)
6 TEMP := $(shell mktemp -d -t tmp.XXXXXXXXXX)
7 TDIR = $(TEMP)/$(PROJECT)
8 VERS = $(shell /bin/date "+%Y-%m-%d---%H-%M-%S")
9 DATE = $(shell /bin/date "+%Y-%m-%d")
10 # Colors
11 RED = \033[0;31m
12 CYAN = \033[0;36m
13 NC = \033[0m
14 echoPROJECT = @echo -e "$(CYAN) <$(PROJECT)>"
15
16 # default
17 all:
18     $(MAKE) article
19     $(MAKE) minimize
20     $(MAKE) zip
21     $(MAKE) count.colorpages
22     $(echoPROJECT) "$(RED) * all files processed * $(NC)"
23
24 # compile article
25 article:
26     $(echoPROJECT) "$(RED) * making article * $(NC)"
27     latexmk -lualatex -quiet -f -cd -view=pdf -output-directory=tmp $(PROJECT).
28     ↪tex
29     @cp tmp/$(PROJECT).pdf .
30     $(echoPROJECT) "$(RED) * article compiled * $(NC)"
31
32 # zip files for sending etc.
33 zip: article
34     $(echoPROJECT) "$(RED) * start zipping files * $(NC)"
35     @-mkdir archive
36     @rm -f archive/$(PROJECT)-$(DATE)*.zip
37     @mkdir $(TDIR)
38     @cp $(PROJECT).{bib,tex,pdf} README.md makefile $(TDIR)

```

```

38 cd $(TEMP); \
39 zip -Drq $(PWD)/archive/$(PROJECT)-$(VERS).zip $(PROJECT)
40 $(echoPROJECT) "$(RED) * files zipped * $(NC)"
41
42 # count pages with colors > https://stackoverflow.com/a/28369599
43 count.colorpages: article
44     $(echoPROJECT) "$(RED) * counting colored pages * $(NC)"
45     @ gs \
46     -o - \
47     -sDEVICE=inkcov \
48     $(PROJECT).pdf \
49     |tail -n +5 \
50     |sed '/^Page*/N;s/\n//'\
51     |sed -E '/Page [0-9]+ 0.00000 0.00000 0.00000 / d' \
52     | tee $(PROJECT).csv
53     @echo -e "Total amount of pages with color: "
54     @ gs -o - -sDEVICE=inkcov $(PROJECT).pdf | \
55     grep -v "^ 0.00000 0.00000 0.00000" | grep "^ " | wc -l
56     $(echoPROJECT) "$(RED) * colored pages counted * $(NC)"
57
58 # minimize PDF
59 minimize: article
60     $(echoPROJECT) "$(RED) * minimizing article * $(NC)"
61     @-mkdir archive
62     @rm -f archive/$(PROJECT)-$(DATE)*.pdf
63     gs \
64     -sDEVICE=pdfwrite \
65     -dCompatibilityLevel=1.4 \
66     -dPDFSETTINGS=/printer \
67     -dNOPAUSE \
68     -dQUIET \
69     -dBATCH \
70     -sOutputFile=archive/$(PROJECT)-$(VERS).pdf \
71     $(PROJECT).pdf
72     $(echoPROJECT) "$(RED) * article minimized * $(NC)"

```

Der Einsatz von *make*

Wie bereits erwähnt, wird das »Ziel« *all* ausgeführt, wenn man im Terminal lediglich *make* eingibt. In unserer Datei werden alle »Ziele« nun standardmäßig ausgeführt.

Möchte man hingegen nur ein bestimmtes `||Ziel||` ausführen, kann man dieses mit `|make <ZIEL>|` direkt ansteuern, beispielsweise `|make zip|`.

Diese `|makefile|`-Datei lässt sich nach Belieben ergänzen und verändern, um auch auf Projekt spezifische Anforderungen zu reagieren.