

Automatisierung mit `make`

PeP et al. Toolbox Workshop



PEP ET AL. E.V.
PHYSIKSTUDIERENDE UND
EHMALIGE PHYSIKSTUDIERENDE
DER TU DORTMUND

2015

Problem:

- kurz vor Abgabe noch neue Korrekturen einpflegen
 - Tippfehler korrigieren, Plots bearbeiten
- $\text{T}_{\text{E}}\text{X}$ ausführen, ausdrucken
- vergessen, Plots neu zu erstellen

Lösung: Make!

- sieht, welche Dateien geändert wurden
- berechnet nötige Operationen
- führt Python-Skript aus, führt T_EX aus

- Automatisierung verhindert Fehler
- Dient als Dokumentation
- Reproduzierbarkeit unverzichtbar in der Wissenschaft
- Idealfall: Eingabe von `make` erstellt komplettes Protokoll/Paper aus Daten
- Spart Zeit, da nur nötige Operationen ausgeführt werden

- Datei heißt Makefile, keine Endung!
 - bei Windows Dateierendungen einschalten!
(<http://support.microsoft.com/kb/865219/de>)
- besteht aus Rules:

Rule

```
target: prerequisites
    recipe
```

target Datei(en), die von dieser Rule erzeugt werden

prerequisites Dateien, von denen diese Rule abhängt

recipe Befehle: prerequisites → target (mit Tab eingerückt)

```
plot.pdf: plot.py data.txt  
python plot.py
```

```
all: report.pdf # convention
```

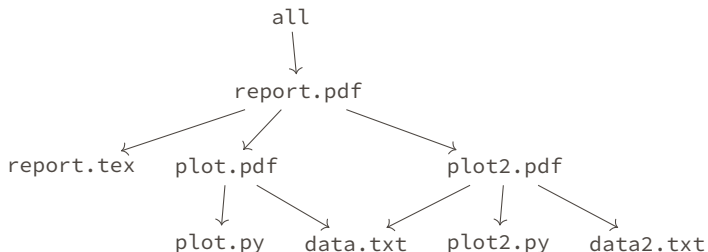
```
plot.pdf: plot.py data.txt  
python plot.py
```

```
report.pdf: report.tex  
lualatex report.tex
```

```
report.pdf: plot.pdf # add prerequisite
```

make eingeben:

- all braucht report.pdf
 - report.pdf braucht plot.pdf
 - python plot.py
 - lualatex report.tex



- Abhängigkeiten bilden einen DAG (directed acyclic graph)
- Dateien werden neu erstellt, falls sie nicht existieren oder älter als ihre Prerequisites sind
- Prerequisites werden zuerst erstellt
- top-down Vorgehen

<code>make target</code>	statt des ersten in der Makefile genannten Targets (meist <code>all</code>) nur <code>target</code> erstellen
<code>make -n</code>	dry run: Befehle anzeigen aber nicht ausführen
<code>make -d</code>	debug: anzeigen, warum make sich so entschieden hat
<code>make -p</code>	Datenbank aller Abhängigkeiten ausgeben

→ Nützlich, wenn man einen Plot bearbeitet: `make plot.pdf`

Es ist eine (nützliche) Konvention, dass `make clean` alles vom `Makefile` erstellte löscht.

```
clean:  
    rm plot.pdf report.pdf
```

Das Projekt sollte dann so aussehen wie vor dem ersten Ausführen von `make`.

build-Ordner: Projekt sauber halten

```
all: build/report.tex
```

```
build/plot.pdf: plot.py data.txt | build  
    python plot.py # savefig('build/plot.pdf')
```

```
build/report.pdf: report.tex build/plot.pdf | build  
    lualatex --output-directory=build report.tex
```

```
build:  
    mkdir -p build
```

```
clean:  
    rm -rf build
```

```
.PHONY: all clean
```

- | build ist ein order-only Prerequisite: Alter wird ignoriert
- Targets, die bei .PHONY genannt werden, entsprechen nicht Dateien (guter Stil)

Mehrere unabhängige Auswertungen: könnte man sie parallel ausführen?

Ja! `make -j4` (4 Prozesse gleichzeitig)

```
plot1.pdf plot2.pdf: plot.py data.txt
python plot.py
```

Problem: `make` führt `plot.py` gleichzeitig zweimal aus

Lösung: manuell synchronisieren

```
plot1.pdf: plot.py data.txt
python plot.py
```

```
plot2.pdf: plot1.pdf
```

Wenn man `plot2.pdf` aber nicht `plot1.pdf` löscht, kann `make` nicht mehr `plot2.pdf` erstellen.