Automatisierung mit `make`

PeP et al. Toolbox Workshop

PeP et al. e.V.

Physikstudierende und
ehemalige Physikstudierende
der TU Dortmund

2017

Problem:

Kurz vor Abgabe noch neue Korrekturen einpflegen

1. Tippfehler korrigieren, Plots bearbeiten
2. TEX ausführen, ausdrucken

→ vergessen, Plots neu zu erstellen

→ zurück zu Schritt 1 ...

Lösung: Make

- → prüft, welche Dateien geändert wurden
- → berechnet nötige Operationen um Abhängigkeiten zu erfüllen
- → führt Befehle aus
  - → Python-Skripe
  - → TEX
  - → etc ...

Motivation

- → **Automatisierung** verhindert Fehler
- → Dient als **Dokumentation**
- → **Reproduzierbarkeit**: unverzichtbar in der Wissenschaft
- → **Spart Zeit**: nur notwendige Operationen werden ausgeführt

Idealfall: Eingabe von `make` erstellt komplettes Protokoll/Paper aus Daten

Makefile

- Von `make` benutzte Datei heißt `Makefile` (keine Endung)
  - bei Windows Dateiendungen einschalten, siehe `http://support.microsoft.com/kb/865219/de`
- `Makefile` besteht aus Regeln (Rules):

Rule

```
target: prerequisites
    recipe
```

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

`prerequisites` Dateien, von denen diese Rule abhängt

`recipe` Befehle, um vom `prerequisites` zu `target` zu kommen
  - wird mit Tab unter `target: prerequisites` eingerückt

Einfachstes Beispiel

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

- Wir wollen `plot.pdf` erzeugen (`target`)
  - `plot.pdf` hängt von `plot.py` und `data.txt` ab (`prerequisites`)
  - Der Befehl, um `plot.pdf` aus den `prerequisites` zu erhalten ist `python plot.pdf`

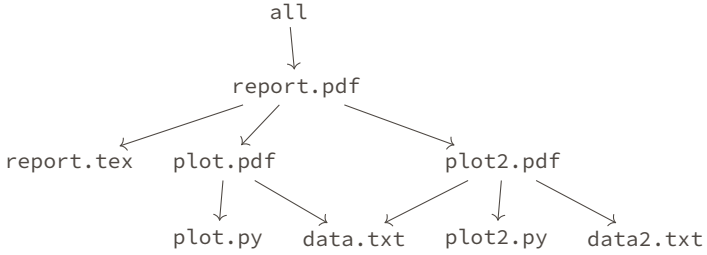
Beispiel

```
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`

Funktionsweise



- 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

Argumente für `make`

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

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

make clean

(Nützliche) Konvention: `make clean` löscht alle vom `Makefile` erstellten Dateien/Ordner.

```
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.pdf

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)

Expert

Können mehrere unabhängige Auswertungen parallel ausgeführt werden?
→ Ja: `make -j4` (nutzt 4 Prozesse gleichzeitig)

Problem: Manchmal führt `make` Skripte gleichzeitig zweimal aus (hier `plot.py`)

```
all: report.txt

report.txt: plot1.pdf plot2.pdf
  touch report.txt

plot1.pdf plot2.pdf: plot.py data.txt
  python plot.py  # plot.py produziert sowohl plot1.pdf als auch plot2.pdf
```

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.