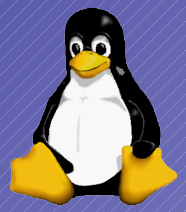
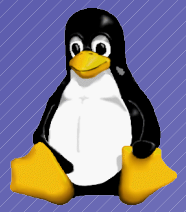
Für Macher

GNU Make
nicht nur für
Programmierer



Überblick

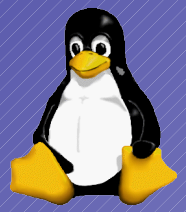
- Automatisches Erstellen von Dateien
- Komplexe Regeln möglich
- Parametrisierbar
- Parallelisierbar



Einfache Regeln

- Erstellen einer Datei “aus dem Nichts” oder aus einer anderen Datei
- Angaben: Quelle, Ziel
- Der Weg ist per Tabulator(!!) eingerückt.

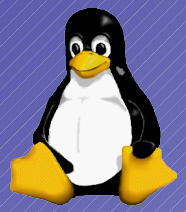
```
#My Makefile

zieldatei: quelldatei
	cat quelldatei >zieldatei

foo:
	touch foo
```



Regelketten

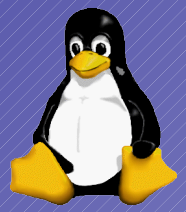
- Make kann allein herausfinden, welche Regeln gebraucht werden

```
#My Makefile

quark: milch
	cat milch >quark
	echo Gerinnung >>quark

milch: kuh
	grep melken kuh >milch

schnitzel: kuh
	cut -f3-3 kuh >schnitzel
```



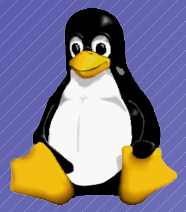
Aufruf

- `make ziel`
  - wenn `ziel` weggelassen: erste Regel im Makefile wird ausgeführt
  - Konvention: erste Regel heißt „all“
- `make -j n`
  - Parallelisieren: max. n Prozesse gleichzeitig
- …



Variablen

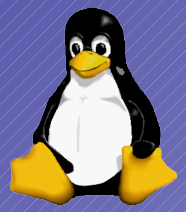
- Machen Makefile parametrisierbar.
- Format: $V oder $(Variable)
- Konvention: Großbuchstaben
- Ein paar sind vordefiniert:
  - $(CC) – C-Compiler
  - $(CXX) – C++-Compiler
  - $(CFLAGS)/$(CXXFLAGS) – Compiler Flags
  - $(LDFLAGS) – Linker-Flags
  - ...



Variablen 2

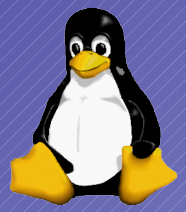
- Zuweisung: `var=text...`
- Erweiterung: `var+=mehr text...`

```
#My Makefile

MELKER = grep
MELKER += melken

quark: milch
	cat milch >quark
	echo Gerinnung >>quark

milch: kuh
	$(MELKER) kuh >milch
```



Spezialvariablen

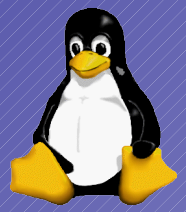
- $@ - Zieldatei
- $< - erste Quelldatei
- $^ - alle Quelldateien
- $? - nur Quelldateien, die neuer sind als die Zieldatei
- $(MAKE) – Befehl für make selbst



Suffix-Regeln

- .ziel.quelle: oder .ziel:
- Suffixe müssen registriert werden

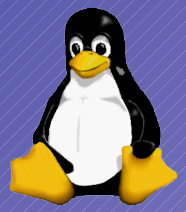
```
#My Makefile

.SUFFIXES: .milch .quark

.milch:
	touch $@

.milch.quark:
	cat $< >$@
	echo Gerinnung >>$@
```



Muster-Regeln

- Flexibler als Suffix-Regeln
- % wird als „Wildcard“ verwendet
- $* enthält den Wildcard-Teil

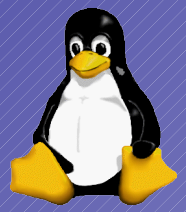
```
#My Makefile

%.milch:
	touch $@

%.quark: %.milch
	cat $< >$@
	echo Gerinnung >>$@

%.milch: kuh_%
	grep melken $< tank_$* >$@
```



Muschelsuppe

- Shell kann für Variablen benutzt werden
- komplexe Shell-Kommandos über mehrere Zeilen mit \
- Lange Abhängigkeiten mit \ verketten
- Shell-$ ist $$

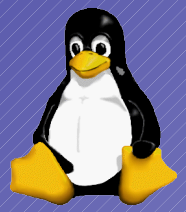
```
BOTTICHE = $(shell echo *.bottich)

kaese: $(BOTTICHE) \
  quark.elsa \
  quark.hanna
      rm -f $@
      for i in $^ ; do \
         cat $$i >> $@ ;\
      done
```



Variablen-Ersetzungen

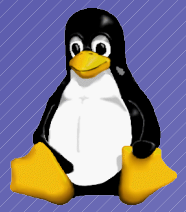
- Endungen können ersetzt werden:
  - $(VAR:.alt=.neu)

```
KUEHE = $(shell echo *.kuh)

kaese: $(KUEHE:.kuh=.quark)
	...

%.milch: %.kuh
	grep melken $<  > $@

%.quark: %.milch
	...
```



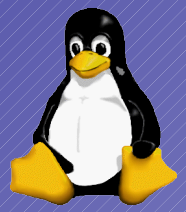
Spezialregeln

- .PHONY: Abhängigkeiten werden immer ausgeführt
- .SECONDARY: löscht diese Dateien nicht, wenn sie Zwischenresultate sind
- .PRECIOUS: löscht auch nicht, wenn make gekillt wird



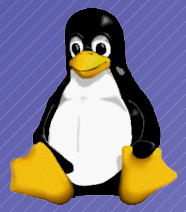
Regel-Konventionen

- all: erste Regel, kompiliert alles
- install: installiert alles
- clean: räumt auf (phony!)
- doc: baut Dokumentation
- test: test-Skripte ausführen



Wie weiter?

```
info make
```



Fragen?

?