Make - Grundkonzept

- *Make* dient im Zusammenhang mit der Programmentwicklung, die bei der Erstellung eines Programmes anfallenden Schritte zu koordinieren.
- Genauer *Make* dient allgemein zur Aktualisierung und Regenerierung von miteinander in Beziehung stehenden Dateien.
- Der Grundgedanke dabei ist, dass gewisse Dateien in Abhängigkeit zu anderen Dateien stehen. So hängen beispielsweise Objektdateien von Ihren jeweiligen Quellcode- und Headerdateien ab.
- Zur Steuerung von make dient ein *Makefile*.
- Diese kümmert sich beispielsweise um den Aufruf des Compilers, des Linkers oder um die Installierung des ausführbaren Programmes.

Make – Regeln

- **Beziehungen zwischen Dateien werden durch Regeln ausgedrückt.**
- **Eine Regel hat die folgende allgemeine Form:**

  Ziel:[:] [Abhängigkeiten] [; [Kommando]]

  Kommando: tab [- @] Kommando

- **Dabei haben Ziel, Abhängigkeiten und Kommandos folgende Bedeutungen:**
  - **Ziel = eine zu erzeugende Datei**
  - **Abhängigkeiten = Dateien, von denen die zu erzeugende Datei abhängt**
  - **Kommandos = Befehlsaufrufe, die zur Erzeugung des Zieles notwendig sind**

Make – Regeln (cont.)

- **Um ein Ziel zu erzeugen, testet *make* dessen Abhängigkeiten und eventuell rekursiv deren Abhängigkeiten.**
- **Das Ziel wird nur neu erzeugt, falls es noch nicht existiert oder seine Abhängigkeiten jünger sind als es selbst.**
- **Letzteres ermittelt *make* anhand des Erzeugungsdatums der betreffenden Dateien.**
- **Beispiel:**

```
hello: hello.o
        gcc -c hello.c
```

Make – Aufruf

- **Der Aufruf erfolgt in der Form:**

  make [-f Makefile] [Option] [Ziel]

- **Option:**
  - **-i = Ignoriere Fehler (der Normalfall ist Programmabbruch).**
  - **-n = Tu nichts, zeige nur die auszuführenden Kommandos an.**
  - **-s = Sei still, zeige die ausgeführten Kommandos nicht an.**
  - **-p = Ausgabe aller Makrodefinitionen, Suffixregeln, Suffixen und Zielregeln**

- **Ziel = entweder ein im *Makefile* beschriebenes Ziel oder aber auch ein leerer String**

Make – Aufruf (cont.)

- **Es können auch Ziele definiert werden, die zwar Abhängigkeiten besitzen, jedoch keine Kommandos.**
- **Nimm an, es gibt neben der *hello* Regel noch eine weitere Regel zur Erzeugung von *goodbye***

```
goodbye: goodbye.o
        gcc -c goodbye.c
```

- **Beide Ziele können als Abhängigkeiten des Zieles *all* definierten werden:**

```
all: hello goodbye
```

- **Dabei besitzt *all* keine Kommandos. Solche Ziele werden auch als Schwindelziele bezeichnet**

Make – Makros

- **Der Sinn von make-Makros ist es, mehrfach benötigte Text-Ausdrücke an einer Stelle im *Makefile* zu definieren, definieren, um so ggf. eine Änderung nur an einer Stelle vornehmen zu müssen**
- **Ein Makro hat die folgende Form:**

  MAKRONAME* = *Wert

- **Dabei haben MAKRONAME und Wert die folgende Bedeutung:**
  - ***MAKRONAME* = Namen des Makros**
  - ***Wert* = Makrowert**

- **Beispiel:**

  SOURCES = hello.c world.c

Make – Build-in Makros

- **Make kennt eine Anzahl so genannter built-in Makros, welche das Verfassen von Regeln erheblich erleichtern**
  - **$@ = Name des Ziels**
  - **$* = Name des aktiven abhängigen Files, welches jünger als das Ziel ist, aber ohne Suffix**
  - **$^ = Liste all vom Ziel abhängigen Files**
  - **$? = Eine Liste von abhängigen Files, die jünger als das Ziel sind**
  - **$< = Name des aktiven abhängigen Files, welches jünger als das Ziel ist**

Make –Makro Zeichenersetzung

- **Wir hatten zuvor das Makro**

  SOURCES = hello.c world.c

- **definiert.**
- **Um das Makro *OBJECTS* zu erstellen, können wir uns mit mit folgender Ersetzungsregel behelfen:**

  OBJECTS = $(SOURCES:.c=.o)

  oder

  OBJECTS = $(SOURCES:%.c=%.o)

- ***$(OBJECTS)* wird nun zu hello.o und world.o ausgewertet**

Make – Beispiel

```
# Variablen für zu verwendende Tools
CC = g++
INC = -I/home/user1/include -I/usr/include/g++
CFLAGS = -Wall -O3 $(INC)
INSTALL = cp

# Variablen für Dateien
EXECUTABLES = HelloWorld
SOURCES = hello.cc world.cc
OBJECTS = $(patsubst %.cc, $(LIBDIR)/%.o, $(SOURCES))

# Variablen für Verzeichnisse
BINDIR = /home/user1/bin
LIBDIR = /home/user1/lib
```

Make – Beispiel

```
#Regeln
#Diese Regel gibt an, wie aus .cc-files .o-files erzeugt werden
    werden

$(LIBDIR)/%.o:  %.cc
        $(CC) -c $(CFLAGS) $< -o $@

all: $(EXECUTABLES)

install:
        $(INSTALL) $(EXECUTABLES) $(BINDIR)

$(EXECUTABLES): $(OBJECTS)
        $(CC) -o $(EXECUTABLES) $(OBJECTS)
clean:
        rm $(LIBDIR)/*.o $(EXECUTABLES)
```