

SOFTWARE ENGINEERING 2

BUILDMANAGEMENT

MAKE

- Make ist ein Werkzeug zum Automatisieren des Buildvorgangs
- Make stammt aus der UNIX-Welt
Benutzung unter Windows z.B. mittels cygwin
- Abhängigkeiten werden abgebildet
- Nur benötigte Kommandos werden ausgeführt

MAKE – Erstes Beispiel

Target

Kommentar

Abhängigkeitsregel

Wichtig: TAB !

Rezept (recipe)

```
# Makefile to make first programm
# Link object file
prog: main.o
    g++ -g main.o -o prog

# Make object file
main.o: main.cpp
    g++ -g -Wall -O -c main.cpp

# Clean up the mess ...
clean:
    /bin/rm -f prog main.o
```

- Target wird erzeugt, indem
 - Abhängigkeiten geprüft werden
 - Gegebenenfalls die Anweisungen (Rezept) ausgeführt werden
- Sind die Abhängigkeiten aktuell, werden die Anweisungen nicht ausgeführt
- Existieren die Abhängigkeiten nicht, wird nach Regeln gesucht, wie diese gebaut werden können → Rekursiv
- Hat eine Regel keine Abhängigkeit, so wird sie immer ausgeführt („Phony Targets“).
- Bei Ausruf des Makefiles wird das angegebene Target gebaut. Ist keines angegeben, so wird das erste gebaut.
- Eine Regel muss nicht unbedingt eine Anweisung haben. Sie kann dazu verwendet werden, andere Regeln aufzurufen.

MAKE – Mehrere Dateien

```
1 # Makefile to make first programm
2
3 # Link object file
4 prog: main.o MyClass.o
5     g++ -g main.o -o prog
6
7 # Make object file
8 main.o: main.cpp
9     g++ -g -Wall -O -c main.cpp
10
11 # Make object file
12 MyClass.o: MyClass.cpp MyClass.h
13     g++ -g -Wall -O -c MyClass.cpp
14
15 # Clean up the mess .|.
16 .PHONY: clean
17 clean:
18     /bin/rm -f prog main.o MyClass.o
```

Abhängigkeit von
Header



MAKE – Vereinfachungen (1)

- Kommandos und Flags als Variablen zusammenfassen
- Dateien in Variablen zusammenfassen

```
1 # Makefile to make first programm
2
3 # Compiler call
4 CC = g++
5
6 # Linker
7 LD = g++
8
9 # Compiler flags
0 CFLAGS = -g -Wall -O
1
2 # Linker flags
3 LDFLAGS =
4
5 # Clean up command
6 RM = /bin/rm -f
7
8 # Object files
9 OBJS = main.o MyClass.o
0
1 # Program executable
2 PROG = prog
3
4 # Link object file
5 $(PROG): $(OBJS)
6     $(LD) $(OBJS) -o $(PROG)
7
8 # Make object file
9 main.o: main.cpp
0     $(CC) $(CFLAGS) -c main.cpp
1
2 # Make object file
3 MyClass.o: MyClass.cpp MyClass.h
4     $(CC) $(CFLAGS) -c MyClass.cpp
5
6 # Clean up the mess ...
7 .PHONY: clean
8 clean:
9     $(RM) $(OBJS) $(PROG)
```

- Generelle Regeln für Dateitypen

```
1 # Makefile to make first programm
2
3 # Compiler call
4 CC = g++
5
6 # Linker
7 LD = g++
8
9 # Compiler flags
10 CFLAGS = -g -Wall -O
11
12 # Linker flags
13 LDFLAGS =
14
15 # Clean up command
16 RM = /bin/rm -f
17
18 # Object files
19 OBJS = main.o MyClass.o
20
21 # Program executable
22 PROG = prog
23
24 # Link object file
25 $(PROG): $(OBJS)
26     $(LD) $(OBJS) -o $(PROG)
27
28 # Make object files
29 %.o: %.cpp
30     $(CC) $(CFLAGS) -c $<
31
32 # Clean up the mess ...
33 .PHONY: clean
34 clean:
35     $(RM) $(OBJS) $(PROG)
```

%.cpp – Wildcard für alle CPP Dateien

%.o – Wildcard für alle Objekt-Dateien

\$< steht für die gefundene Abhängigkeit
(hier: <Datei>.cpp)

Ähnlich:

\$@ Targetname

\$* Ersetzung %

- Einsatz von Skripten beim Einstellen ins KM
- Prüfung auf Compilierfähigkeit
- Automatisches Starten von
 - Tests
 - Dokumentation
- Buildserver
 - Automatisches Bauen eines kompletten SW-Standes
 - Zyklisch
 - Ereignisgetrieben
 - Dokumentation der „Build-Historie“
 - Bereitstellung der SW