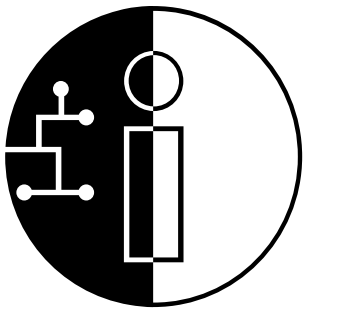
GNU-Make

Daniel Otte
(daniel.otte@rub.de)



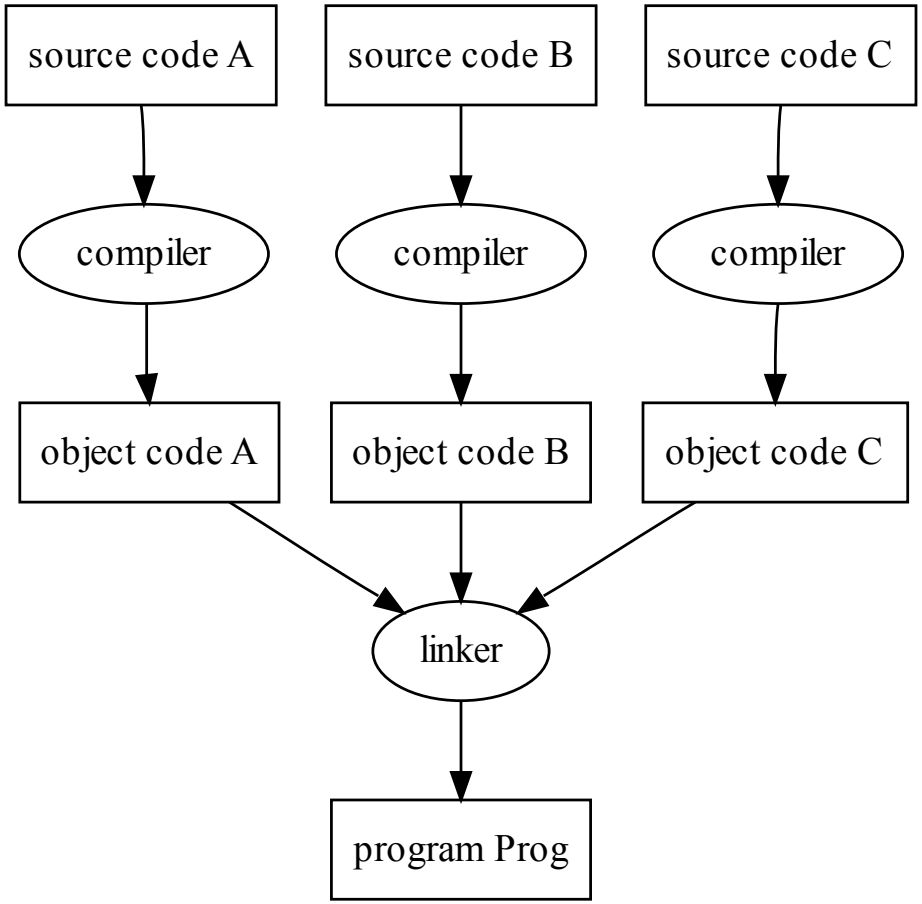
Labor Projekttage
13.-16. November 2008

Beispiel Programm

Ein einfaches Programm aus 3 Modulen

Beispiel Programm

Ein einfaches Programm aus 3 Modulen

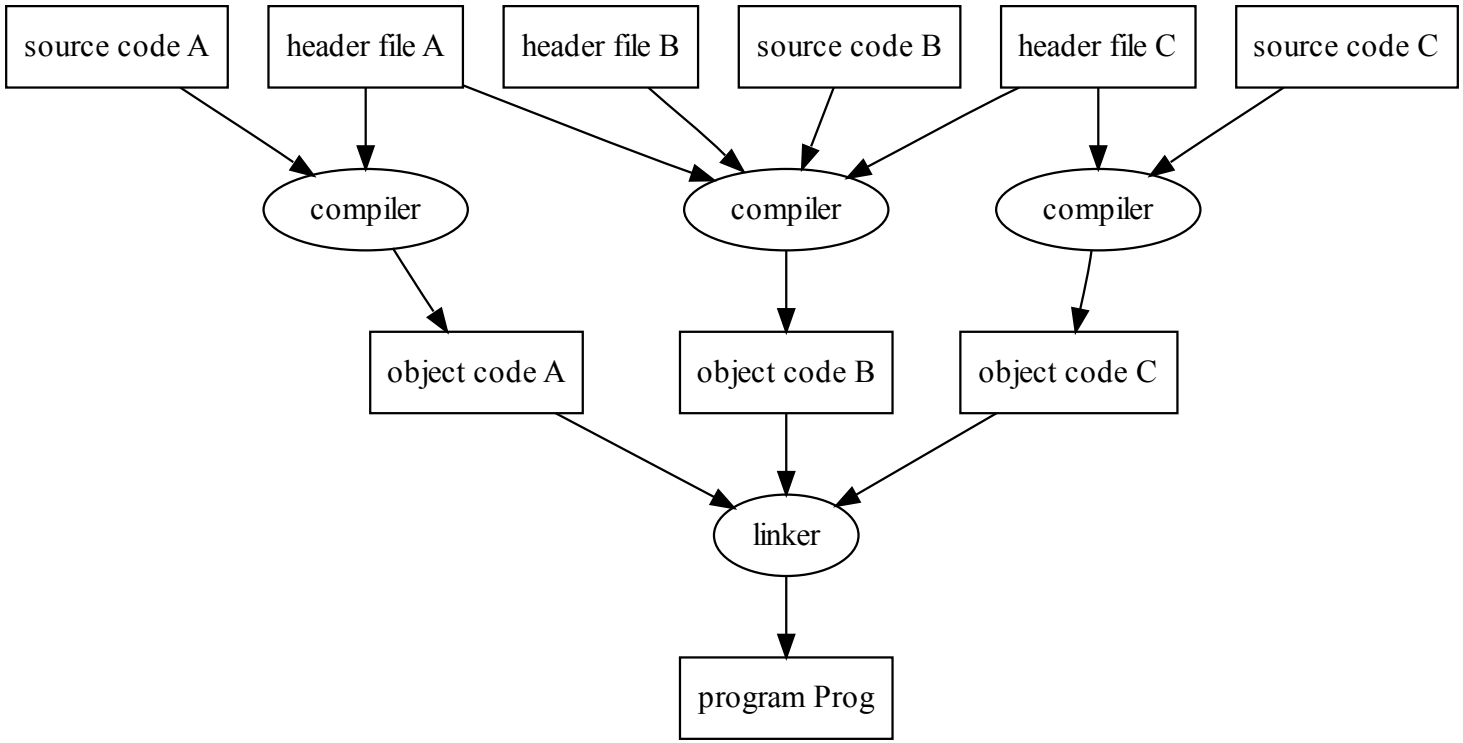


Beispiel Programm2

Ein anderes einfaches Programm aus 3 Modulen, mit mehr Abhängigkeiten

Beispiel Programm2

Ein anderes einfaches Programm aus 3 Modulen, mit mehr Abhängigkeiten

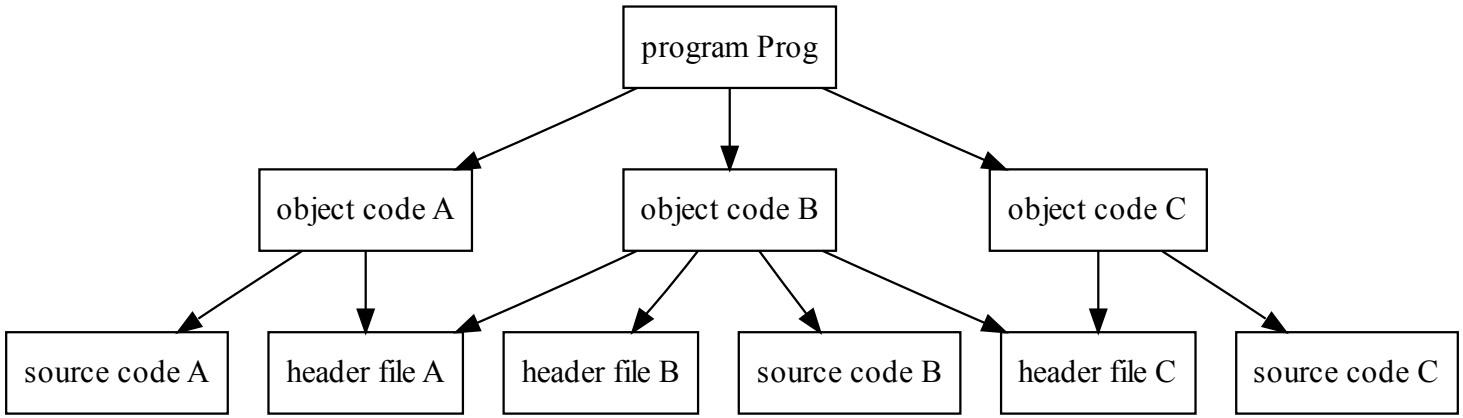


Abhängigkeiten Programm2

Abhängigkeiten in Programm2

Abhängigkeiten Programm2

Abhängigkeiten in Programm2

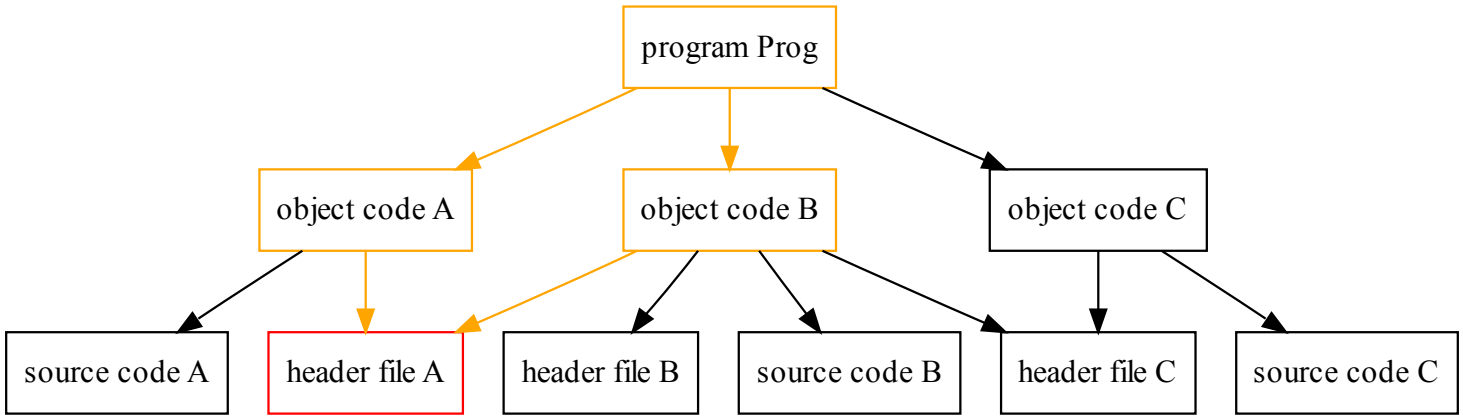


Abhängigkeiten Programm2

Folgen einer Änderung in *header file A*

Abhängigkeiten Programm2

Folgen einer Änderung in *header file A*



Was Make für dich tut

Was Make für dich tut

- Prüfung des Dateialters (mit Hilfe des Timestamps des Dateisystems)

Was Make für dich tut

- Prüfung des Dateialters (mit Hilfe des Timestamps des Dateisystems)
- (neu) Erzeugen der Dateien, deren Abhängigkeiten sich geändert haben

Was du tun must

Was du tun must

- Make die Abhängigkeitsverhältnisse mitteilen (welche Datei hängt von wecher ab)

Was du tun must

- Make die Abhängigkeitsverhältnisse mitteilen (welche Datei hängt von wecher ab)
- Make mitteilen wie die Dateien zu bauen sind

Aufruf

Normalerweise wird dem Make-Aufruf die Liste der Ziele (targets) übergeben, dabei kann es sich handeln um:

Wenn kein Target angegeben wird ist der Default *all*. Die Dokumentation der Optionen findest du in der Manpage.

Aufruf


```
make [..opts..] [target] {target}
```

Normalerweise wird dem Make-Aufruf die Liste der Ziele (targets) übergeben, dabei kann es sich handeln um:

Wenn kein Target angegeben wird ist der Default *all*. Die Dokumentation der Optionen findest du in der Manpage.

Aufruf


```
make [..opts..] [target] {target}
```

Normalerweise wird dem Make-Aufruf die Liste der Ziele (targets) übergeben, dabei kann es sich handeln um:

- Dateien die zu erstellen sind, oder

Wenn kein Target angegeben wird ist der Default *all*. Die Dokumentation der Optionen findest du in der Manpage.

Aufruf


```
make [..opts..] [target] {target}
```

Normalerweise wird dem Make-Aufruf die Liste der Ziele (targets) übergeben, dabei kann es sich handeln um:

- Dateien die zu erstellen sind, oder
- spezielle Targets die Instruktionen enthalten (Bsp. *clean*)

Wenn kein Target angegeben wird ist der Default *all*. Die Dokumentation der Optionen findest du in der Manpage.

Beispiel 1

Ein einfaches Beispiel:

```
1  all: programm

3  programm: code_a.o code_b.o code_c.o
       gcc -o programm code_a.o code_b.o code_c.o
5
   code_a.o: code_a.c code_a.h
7      gcc -c -o code_a.o code_a.c
   code_b.o: code_b.c code_b.h code_a.h code_c.h
9      gcc -c -o code_b.o code_b.c
   code_c.o: code_c.c code_c.h
11     gcc -c -o code_c.o code_c.c
```

Beispiel 2

Ein einfaches Beispiel mit automatischen Variablen:

```
all: programm

programm: code_a.o code_b.o code_c.o
	gcc -o $@ $^

code_a.o: code_a.c code_a.h
	gcc -c -o $@ $<
code_b.o: code_b.c code_b.h code_a.h code_c.h
	gcc -c -o $@ $<
code_c.o: code_c.c code_c.h
	gcc -c -o $@ $<
```

Variable	Wert
$@	target
$^	Abhängigkeiten
$<	erste Abhängigkeit

Beispiel 3

Ein einfaches Beispiel mit generischen Regeln:

```
all: programm

programm: code_a.o code_b.o code_c.o
	gcc -o $@ $^

code_a.o: code_a.c code_a.h
code_b.o: code_b.c code_b.h code_a.h code_c.h
code_c.o: code_c.c code_c.h

%.o: %.c
	gcc -c -o $@ $<
```

Beispiel 4

Ein einfaches Beispiel mit Variablen:

```
CC = gcc

all: programm

programm: code_a.o code_b.o code_c.o
	$(CC) -o $@ $^

code_a.o: code_a.c code_a.h
code_b.o: code_b.c code_b.h code_a.h code_c.h
code_c.o: code_c.c code_c.h

%.o: %.c
	$(CC) -c -o $@ $<
```

variable = *wert*	Zuweisung
$(*variable*)	Referenzierung

It’s all about ...

It’s all about ...

- Strings

It’s all about ...

- Strings
- Lists (of Strings)

Listen sind ...

Listen sind:

Listen sind ...

Listen sind:

- Strings, die Elemente sind durch Whitespace getrennt

subst

$(subst from,to,text)

Führt Stringersetzung durch.

Beispiel:
$(subst, ee,EE,speed tree) $\longrightarrow$ 'spEEd trEE'

patsubst

$(patsubst pattern,replacement,text)

Führt Musterersetzung durch.

Beispiel:
$(patsubst, %.c,%.d,code.c code.h edit.c) $\longrightarrow$ 'code.d code.h edit.d'

strip

$(strip string)

Entfernt Whitespace am Anfang und am Ende

Beispiel:
$(strip, code.c code.h edit.c) $\longrightarrow$ 'code.d code.h edit.d'

findstring

$(findstring find,in)

Sucht nach *find* in *in* und evaluiert zu *find* wenn gefunden sonst zu ’ ’ (empty).

Beispiel:
$(findsring, a, a b c) $\longrightarrow$ ’a’

filter

$(filter pattern ...,text)

Evaluiert zu einer Liste von Strings aus *text* die alle auf eins (oder mehere) der *pattern* passen.

Beispiel:
$(filter, %.c %.s, a.c b.h c.s) $\longrightarrow$ 'a.c c.s'

filter-out

$(filter-out pattern ...,text)

Evaluiert zu einer Liste von Strings aus *text* die alle auf keins der *pattern* passen.

Beispiel:
$(filter-out, %.c %.s, a.c b.h c.s) $\longrightarrow$ 'b.h'

sort

$(sort list)

Sortiert die Liste und entfernt doppelte Einträge.

Beispiel:
$(sort, foo bar foo lose) $\longrightarrow$ 'bar foo lose'

word

$(word n, text)

Evaluiert zu dem n-ten String aus Text (Indizierung beginnend bei 1).

Beispiel:
$(word 2, foo bar foo lose) $\longrightarrow$ 'bar'

words

$(words text)

Evaluiert zu der Anzahl von Strings in *text*.

Beispiel:
$(words foo bar foo lose) $\longrightarrow$ '4'

firstword

$(firstwordword names ...)

Evaluiert zu dem ersten String in *names*.

Beispiel:
$(firstword foo bar foo lose) $\longrightarrow$ 'foo'

lastword

$(lastwordword names ...)

Evaluiert zu dem letzten String in *names*.

Beispiel:
$(lastword foo bar foo lose) $\longrightarrow$ 'lose'

End Of Presentation