UnixTools Vortrag

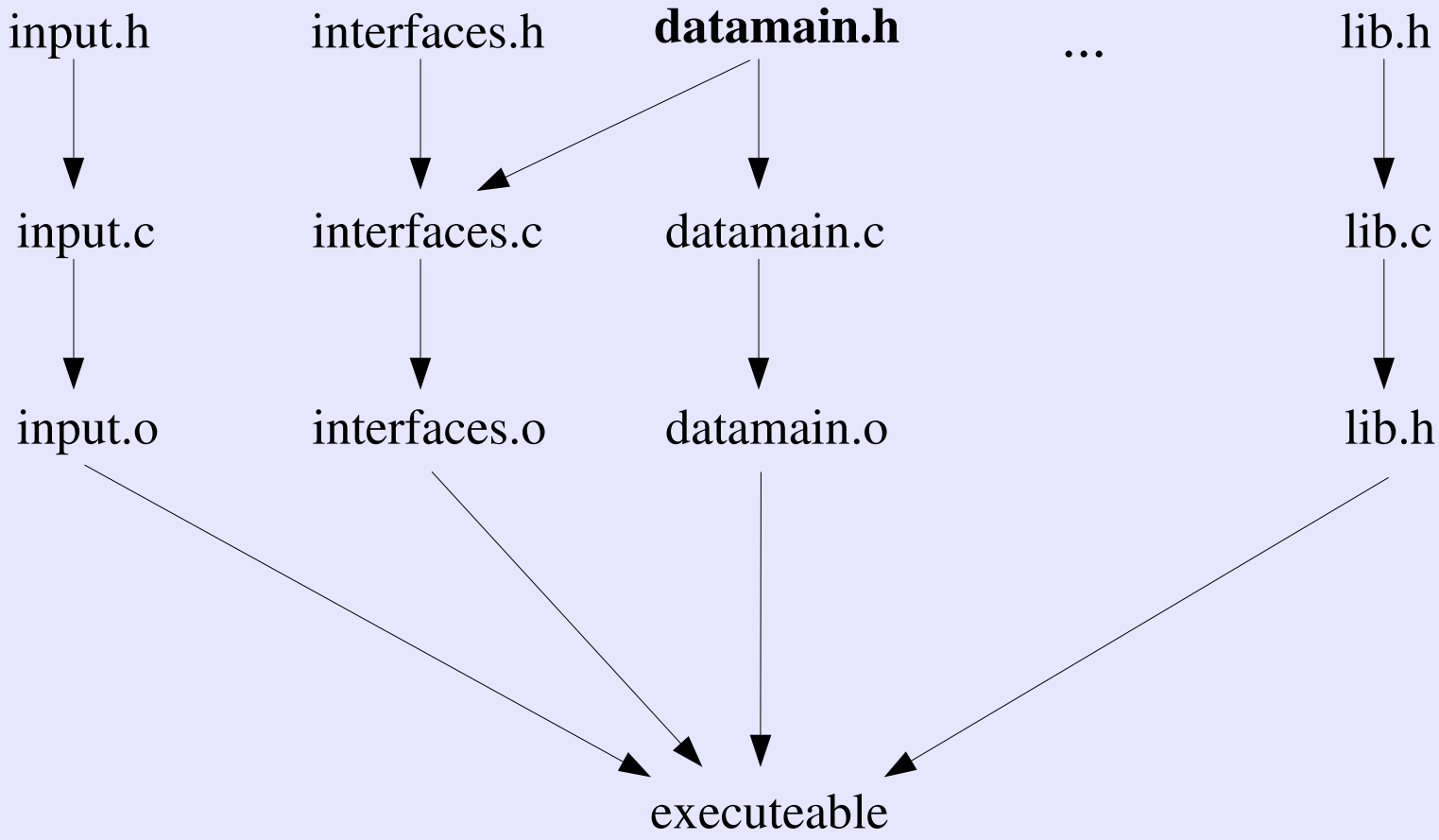
make

Markus Graßl

Gliederung

- **Einleitung**
- **Aufbau eines makefiles**
- **Inhalt eines makefiles**
- **Errors**
- **Resumee**

Abhaengigkeiten



make

- Programm zum compilieren von Dateien

make

- Programm zum compilieren von Dateien
- Kein eigenstaendiger Compiler

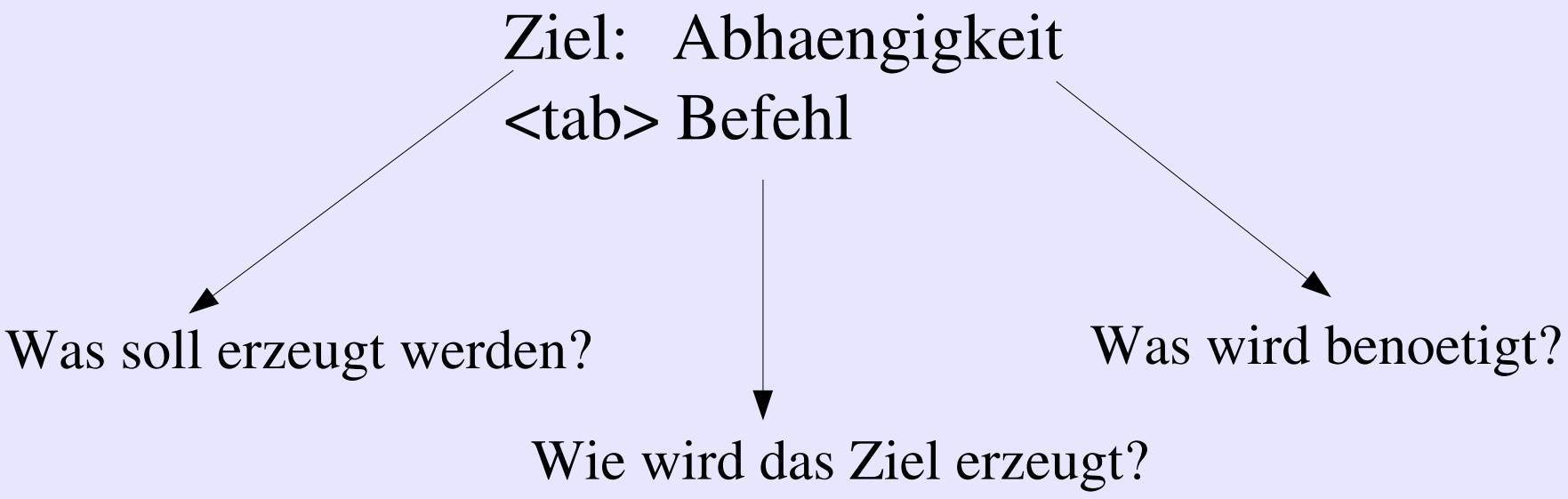
make

- Programm zum compilieren von Dateien
- Kein eigenstaendiger Compiler
- Vereinfachung des Compiliervorgangs

make

- Programm zum compilieren von Dateien
- Kein eigenstaendiger Compiler
- Vereinfachung des Compiliervorgangs
- Vielseitig einsetzbar (programmiersprachen-unabhaengig)

Aufbau eines makefiles



Aufbau eines makefiles

Ziel:

- Name des Programm(moduls), dass durch die nachfolgenden Kommandos generiert wird

Aufbau eines makefiles

Ziel:

- Name des Programm(moduls), dass durch die nachfolgenden Kommandos generiert wird

Abhaengigkeit:

- Hier stehen die Namen der Dateien und Ziele, von denen das Ziel abhängt.

Aufbau eines makefiles

Ziel:

- Name des Programm(moduls), dass durch die nachfolgenden Kommandos generiert wird

Abhaengigkeit:

- Hier stehen die Namen der Dateien und Ziele, von denen das Ziel abhängt.

Befehl:

- Hier stehen die Aktionen, die im Falle einer erfüllten Abhängigkeit auszuführen sind.

Aufbau eines makefiles

Ziel:

- Name des Programm(moduls), dass durch die nachfolgenden Kommandos generiert wird

Abhaengigkeit:

- Hier stehen die Namen der Dateien und Ziele, von denen das Ziel abhängt.

Befehl:

- Hier stehen die Aktionen, die im Falle einer erfüllten Abhängigkeit auszuführen sind.
- Jede Aktion **muss** auf einer eigenen Zeile stehen und durch einen Tabulator eingerückt sein (**keine Leerzeichen!!!**).

Aufbau eines makefiles

Ziel:

- Name des Programm(moduls), dass durch die nachfolgenden Kommandos generiert wird

Abhaengigkeit:

- Hier stehen die Namen der Dateien und Ziele, von denen das Ziel abhängt.

Befehl:

- Hier stehen die Aktionen, die im Falle einer erfüllten Abhängigkeit auszuführen sind.
- Jede Aktion **muss** auf einer eigenen Zeile stehen und durch einen Tabulator eingerückt sein (**keine Leerzeichen!!!**).
- Sehr lange Zeilen können umgebrochen werden, indem am Ende der Zeile ein »\« angefügt wird.

Beispiel eines makefiles

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o
main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```

Inhalt eines makefiles

- Explizite Regeln
- Implizite Regeln
- Variablen / Makros
- Anweisungen
- Kommentare

Inhalt eines makefiles

Explizite Regeln:

- Eine explizite Regel gibt für **genau eine** Datei an, von welchen anderen Dateien sie abhängt.

Inhalt eines makefiles

Explizite Regeln:

- Eine explizite Regel gibt für **genau eine** Datei an, von welchen anderen Dateien sie abhängt.
- Sie gibt genau an, mit welchem/n Kommando(s) sie aktualisiert werden kann.

Inhalt eines makefiles

Explizite Regeln:

- Eine explizite Regel gibt für **genau eine** Datei an, von welchen anderen Dateien sie abhängt.
- Sie gibt genau an, mit welchem/n Kommando(s) sie aktualisiert werden kann.

Beispiel:

```
main.o : main.c defs.h
        gcc -c main.c
command.o : command.c defs.h command.h
        gcc -c command.c
```

Inhalt eines makefiles

Implizite Regeln:

- Eine implizite Regel enthält kein konkretes, zu aktualisierendes Ziel, sondern ist ein Schablone zur Erzeugung einer Datei mit einer bestimmten Endung aus einer Datei mit einer anderen Endung.

Inhalt eines makefiles

Implizite Regeln:

- Eine implizite Regel enthält kein konkretes, zu aktualisierendes Ziel, sondern ist ein Schablone zur Erzeugung einer Datei mit einer bestimmten Endung aus einer Datei mit einer anderen Endung.
- Allgemeine implizite Regeln sind in make vordefiniert.

Inhalt eines makefiles

Implizite Regeln:

- Eine implizite Regel enthält kein konkretes, zu aktualisierendes Ziel, sondern ist ein Schablone zur Erzeugung einer Datei mit einer bestimmten Endung aus einer Datei mit einer anderen Endung.
- Allgemeine implizite Regeln sind in make vordefiniert.
- Um implizite Regeln selbst zu definieren benutzt man “pattern rules”

Inhalt eines makefiles

Implizite Regeln:

- Eine implizite Regel enthält kein konkretes, zu aktualisierendes Ziel, sondern ist ein Schablone zur Erzeugung einer Datei mit einer bestimmten Endung aus einer Datei mit einer anderen Endung.
- Allgemeine implizite Regeln sind in make vordefiniert.
- Um implizite Regeln selbst zu definieren benutzt man “pattern rules”.

Beispiel:

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

Inhalt eines makefiles

Variablen / Makros:

- Eine Variable repraesentiert einen “String”, der im Makefile definiert und oefters benutzt wird. (Dateinamen, Compiler-Optionen, usw.)

Inhalt eines makefiles

Variablen / Makros:

- Eine Variable repraesentiert einen “String”, der im Makefile definiert und oefters benutzt wird. (Dateinamen, Compiler-Optionen, usw.)
- Eine Variable besteht aus einer beliebigen Anzahl an Zeichen,
  nicht aber: ':', '#', '=', ' '

Inhalt eines makefiles

Variablen / Makros:

- Eine Variable repraesentiert einen “String”, der im Makefile definiert und oefters benutzt wird. (Dateinamen, Compiler-Optionen, usw.)
- Eine Variable besteht aus einer beliebigen Anzahl an Zeichen,
  nicht aber: ':', '#', '=', ' '
- Viele Variablen sind in make schon vorbelegt (z.B. CC: Compiler für C-Quelltexte, CFLAGS: Compilerargumente usw.)

Inhalt eines makefiles

Beispiel:

```
CC = gcc                                                    #Definition
CFLAGS = -c
TARGET = output
OBJECTS = main.o kbd.o command.o display.o insert.o
search.o files.o utils.o

$(TARGET) : $(OBJECTS)

   $(CC) -o output $(OBJECTS)

main.o : main.c defs.h
   $(CC)  $(CFLAGS)  main.c
kbd.o : kbd.c defs.h
   $(CC)  $(CFLAGS)  kbd.c

usw.

clean:
   rm output  $(OBJECTS)
```

Inhalt eines makefiles

Anweisungen:

- Eine Anweisung bedeutet, fuer das makefile das etwas besonderes zu tun ist, wie z.B. ein anderes makefile einlesen oder einen Teil des makefiles zu ignorieren.

Inhalt eines makefiles

Anweisungen:

- Eine Anweisung bedeutet, fuer das makefile das etwas besonderes zu tun ist, wie z.B. ein anderes makefile einlesen oder einen Teil des makefiles zu ignorieren.

Kommentare:

- Kommentare beginnen in makefiles mit #

Inhalt eines makefiles

make clean:

- Zum Löschen aller überflüssigen, oder aus den Quelldateien erzeugbaren Dateien wird in einem makefile ein Ziel mit dem Namen clean geschrieben

Inhalt eines makefiles

make clean:

- Zum Löschen aller überflüssigen, oder aus den Quelldateien erzeugbaren Dateien wird in einem makefile ein Ziel mit dem Namen clean geschrieben
- Mittels dem Befehl “make clean” kann man das Arbeitsverzeichnis auf das Wesentliche reduzieren.

Inhalt eines makefiles

make clean:

- Zum Löschen aller überflüssigen, oder aus den Quelldateien erzeugbaren Dateien wird in einem makefile ein Ziel mit dem Namen clean geschrieben
- Mittels dem Befehl “make clean” kann man das Arbeitsverzeichnis auf das Wesentliche reduzieren.

Beispiel:

```
clean :
        rm edit main.o kbd.o command.o display.o
insert.o search.o files.o utils.o
```

Errors

Einige Fehlermeldungen:

- `missing separator. Stop.' : ein <TAB> oder eine andere Trennung (':', '=') fehlt

Errors

Einige Fehlermeldungen:

- `missing separator. Stop.' : ein <TAB> oder eine andere Trennung (':', '=') fehlt
- `No rule to make target `xxx'.': Regel, um ein Ziel zu erstellen, fehlt

Errors

Einige Fehlermeldungen:

- `missing separator. Stop.' : ein <TAB> oder eine andere Trennung (':', '=') fehlt
- `No rule to make target `xxx'.': Regel, um ein Ziel zu erstellen, fehlt
- `warning: overriding commands for target `xxx": Mehrere Regeln fuer ein Ziel vorhanden (letzte gefundene Regel wird verwendet)

Errors

Einige Fehlermeldungen:

- `missing separator. Stop.' : ein <TAB> oder eine andere Trennung (':', '=') fehlt
- `No rule to make target `xxx'.': Regel, um ein Ziel zu erstellen, fehlt
- `warning: overriding commands for target `xxx": Mehrere Regeln fuer ein Ziel vorhanden (letzte gefundene Regel wird verwendet)
- `missing target pattern. Stop.": Kein Makro fuer gefundenes Pattern vorhanden

Resumee

- make ist vor allem bei groeßeren Projekten Pflicht!!!

Resumee

- make ist vor allem bei groeßeren Projekten Pflicht!!!
- make ist einfach zu bedienen

Resumee

- make ist vor allem bei groeßeren Projekten Pflicht!!!
- make ist einfach zu bedienen
- make ist vielseitig und sehr maechtig

FIN