

Informatik 2/A

Makefiles

make und Makefiles

- make ist Build-Management-Tool für C und C++ (exist. seit 1976)
- hauptsächlich genutzt, wenn Software aus vielen verschiedenen Sourcefiles besteht
- einzelne Schritte, die sonst auf Konsole manuell gemacht werden müssten (z.B. übersetzen, linken), können so automatisiert werden
- sehr flexibel
- ursprünglich entwickelt für Linux, Unix (u.ä. Systeme)
- heutzutage existieren mehrere Derivate (z.B. cmake, nmake, ...). Größtenteils kompatibel untereinander
- alle notwendigen Informationen in einer Datei (namens Makefile) hinterlegt

Aufbau eines Makefiles

- einfaches Textfile mit Namen *Makefile* (ohne Endung)
- wenn gewünscht, kann Name auch abweichen
- Makefile dient dazu, make mitzuteilen, was getan werden soll (= das Target) und wie (= die zum Target gehörende Regel)
- ausgeführt durch Befehl *make* auf der Konsole
 - ohne Angabe von Argumenten wird make mit default Einstellungen ausgeführt
 - falls gewünscht, können weitere Argumente zum ausführen mitgegeben werden, um z.B. spezielle Targets laufen zu lassen
- Syntax: <Targetname>: <Abhängigkeiten>

gcc <was soll wie übersetzt werden, mit tab eingerückt>

```
targets: prerequisites
command
command
command
```

Aufbau eines Makefiles

- erstes Target (= meistens erste Zeile) ist immer default-target (Hauptziel)
- Falls gewünschtes Target nicht existiert oder Sourcefile neuer ist als Targetfile, wird Target ausgeführt
- Target: *all*
 - Falls man mehrere Targets hat und alle laufen lassen möchte
 - sollte an erster Stelle stehen
 - ist häufig das default Argument
- Target: *clean*
 - häufig benutzt als Target, um Output anderer Targets zu löschen
 - ist jedoch kein spezielles Wort in make
 - ist nicht das erste Target und hat keine Abhängigkeiten
 - d.h. wird nur ausgeführt, wenn man es explizit aufruft mittels `make clean`
 - es sollte keine Datei mit diesem Namen geben

Typischer Durchlauf eines Makefiles

- erstes bzw default Target wird ausgewählt
- falls Abhängigkeiten existieren, werden entsprechende Targets gesucht und ausgewählt, solange bis Target ohne Abhängigkeiten gefunden wurde
- die restlichen Abhängigkeiten werden ausgeführt

```
some_file: other_file
    echo "This will always run, and runs second"
    touch some_file

other_file:
    echo "This will always run, and runs first"
```

einfaches Beispiel

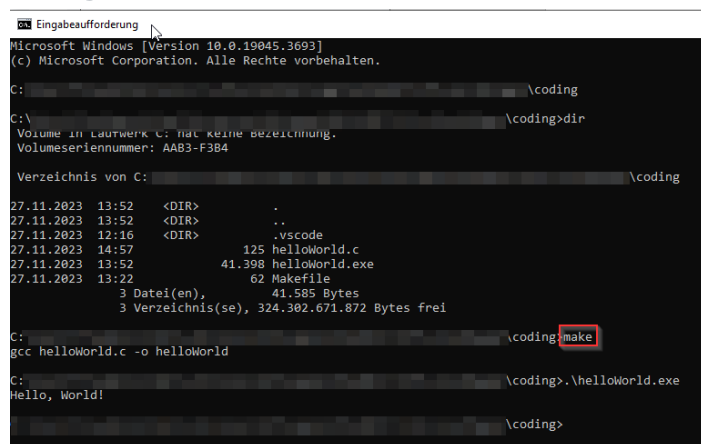
- einfache Datei, die helloWorld-Programm enthält
- Makefile enthält 2 Zeilen

```
helloWorld.c > ...
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

```
M Makefile
1  helloWorld: helloWorld.c
2  |          | gcc helloWorld.c -o helloWorld
3
```

- Konsole öffnen, in entsprechenden Ordner wechseln und make eintippen
➔ helloWorld.exe wird automatisch gebaut



```
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\> cd \coding

C:\coding> dir
Verzeichnis von C:\coding

27.11.2023  13:52    <DIR>          .
27.11.2023  13:52    <DIR>          ..
27.11.2023  12:16    <DIR>          .vscode
27.11.2023  14:57             125 helloWorld.c
27.11.2023  13:52             41.398 helloWorld.exe
27.11.2023  13:22              62 Makefile
             3 Datei(en),    41.585 Bytes
             3 Verzeichnis(se), 324.302.671.872 Bytes frei

C:\coding> make
gcc helloWorld.c -o helloWorld

C:\coding> .\helloWorld.exe
Hello, World!

C:\coding>
```

etwas komplexeres Beispiel

- Dateien
 - main.c
 - namesarchive.c
 - namesarchive.h
- namesarchive soll als dynamische Laufzeitbibliothek nutzbar sein
- ohne Makefile
 - alles manuell in Konsole eintippen
- mit Makefile
 - nur make eingeben, Rest läuft automatisch

```
>gcc -c -Wall main.c
>gcc -Wall -fpic -c namesarchive.c
>gcc -shared -o libnamesarchive.so namesarchive.o
>gcc -Wall -o namesarchive main.o libnamesarchive.so
```

```
>make
gcc -c -Wall main.c
gcc -Wall -fpic -c namesarchive.c
gcc -shared -o libnamesarchive.so namesarchive.o
gcc -Wall -o namesarchive main.o libnamesarchive.so
```

Makefiles in CLion nutzen

- (neuen) Ordner mit allen benötigten Dateien erstellen
- Makefile erzeugen
- Ordner in CLion als Projekt öffnen
- Makefile wird erkannt, Frage nach "soll Target clean ausgeführt werden?"
- Falls Fehler: No rule to make target 'all'

No compilation commands found

- Auf Settings > Build, Execution, Deployment klicken
- bei Build target entweder Namen des gewünschten Targets eingeben (z.B. namesarchive) oder leer lassen (per default steht all drin)
- bei Build auf "reload makefile project" klicken
- Danach alle Settings vorhanden

Makefiles in VSCode nutzen

- make mittels chocolatey systemweit installieren (Windows)

- PowerShell mit Administratorrechten starten
- folgenden Befehl ausführen

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072;  
iex ((New-Object  
System.Net.WebClient).DownloadString('https://community.chocolat  
ey.org/install.ps1'))
```

- danach PowerShell schließen und neu öffnen und folgendes eingeben:
choco install make

- in VSCode im Terminal mit Befehl "make" testen (VSCode evtl nochmal neustarten)

weiterführende Links

- <https://www.c-howto.de/tutorial/makefiles/>
- https://www.gnu.org/software/make/manual/html_node/index.html#SEC_Contents
- <https://makefiletutorial.com/>
- Cheat Sheet: <https://cheatography.com/bavo-van-achte/cheat-sheets/gnumake/>

Autoren / Impressum

- **Autoren**

Prof. Dr.-Ing. Jan Paulus, Prof. Dr. Enrico Schröder, Prof. Dr. Anja Freudenreich

- **Impressum**

Prof. Dr. Anja Freudenreich
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik,
Wassertorstraße 10
904489 Nürnberg, Germany
E-mail : anja.freudenreich@th-nuernberg.de

Dieses Skriptum ist nur für den eigenen Gebrauch im Studium gedacht. Eine Weitergabe ist nur mit Zustimmung des Autors gestattet.