Softwaretechnik Übung: make

Klaus Kusche

1. Wir verwenden GNU make.
   Grundlage der folgenden Übung ist die Online-Doku von GNU **make**:
   http://www.gnu.org/software/make/manual/html_node/index.html

   sowie (kürzer und leichter verdaulich) das Selflinux-**make**-Tutorial:
   http://selflinux.org/selflinux/html/make.html
2. Schreib im Verzeichnis der vorigen Übung einen **Makefile**, um das Programm schrittweise zu bauen (d.h. nicht ein Compiler-Aufruf mit allen Sourcen und dem Executable als Output, sondern ein Aufruf pro Source, um den jeweiligen Object-File zu erzeugen, und ein Aufruf, um aus allen Objects das Executable zu linken).

   Achte dabei auf richtige Dependencies (welcher Source braucht welche Header?).
3. Ruf **make** zweimal auf. Passiert das Erwartete (beim zweiten Mal nichts!)?

   Ändere einen Source- oder Header-File bzw. lösche einen Object-File.
   Passiert beim nächsten **make** das Richtige?
4. Experimentiere mit verschiedenen Arten von Regeln (eine Regel mit Befehl pro Abhängigkeit, eine allgemeine Regel für alle Compiler-Aufrufe, ...).
5. Mach eine Regel für den Doxygen-Aufruf und den LaTex/PDF-**make**-Aufruf.
   Mach eine Regel für das Pseudo-Target **all** (Executable + Dokumentation)
   und eine für **make clean** (alle automatisch erzeugten Files + Dir's löschen).

   Wohin musst du deine **all**-Regel schreiben, damit "**all**" erzeugt wird,
   wenn man nur **make** ohne Angabe eines Targets aufruft?
6. Führe Variablen ein (z.B. für die Header-Files oder die Objects).
7. Bau deine Compile-Regel so um, dass nicht der Compiler-Aufruf angezeigt wird, sondern nur "**Compiling** *filename*". Du wirst dir dafür anschauen müssen, wie du mehrzeilige Befehle angibst und die Ausgabe des Befehls selbst unterdrückst.
8. Aufrufe von **make** (die Man-Page von **make** hilft!):
   - Erzeuge einen Debug-Build, *ohne* den Makefile zu ändern (indem du die Variable für die Compiler-Optionen direkt im **make**-Aufruf angibst).
   - Wie baut man *alles* neu, unabhängig vom Filedatum?
   - Wie lässt man sich anzeigen, was **make** tun würde, ohne es wirklich zu tun?

     Kannst du dabei auch einen geänderten File vortäuschen?
     ("Was müsste alles neu gebaut werden, wenn ich diesen File angreife?")
   - Lass dir alle vordefinierten Regeln und Variablen anzeigen,
     such dir im Ergebnis die C-Compile-Regel heraus.
9. Experimentiere mit **gcc -M** und **makedepend** .

10. Klone dir von **https://github.com/tcsh-org/tcsh** die **tcsh**-Sourcen, konfiguriere sie (gib dabei ein Verzeichnis unterhalb deines Home-Verzeichnisses als Installationspfad an!), baue sie und installiere sie.

    Tipp: **./configure** ..., **make** und **make install**

11. Wie bringst du **make** dazu, max. 5 Compiler zugleich zu starten?
    Bringt das zeitlich etwas, wenn du in Punkt 10 alles neu baust?

    Tipps:
    - Mit **make clean** machst du nach einem Build wieder sauber.
    - Mit **time** ... kannst du die Laufzeit eines Befehls messen.