TGL
Volume 00, Number 0, Pages 000–000
S 0000-0000(XX)0000-0

EXTRA CLANG TOOLS - EMPFEHLUNGEN

Abstract. Empfehlungen zur Nutzung von Clang Tools. Dieses Dokument erfüllt nicht die Bedingungen eines Standards.
Es kann Fehler enthalten.
Alle Angaben ohne Gewähr.
Fehler bitte melden.

1. Allgemeines Setup

LLVM in einer aktuellen Version muss installiert sein. Alle Dokumentation kann unter https://clang.llvm.org/extra/ und https://clang.llvm.org/docs/ClangTools.html gefunden werden.
Unter clang-tools-extra sind alle Tools die in der Entwicklung sind.
Alle für die echte Welt bestimmten Tools sind im normalen clang.
LLVM harmoniert insbeondere auch mit Ninja.
Konkret:
`/build` Unterverzeichnis anlegen. `CMAKE-EXPORT-COMPILE-COMMANDS=ON` und zusätzlich empfehlenswert `CMAKE-C-COMPILER=/path/to/clang` und `CMAKE-CXX-COMPILER=/path/to/clang++` in CMakeLists.txt setzen.

2. Clang-Include-Fixer

Dieses Tool kann fehlende `#include` Direktiven nachrüsten.

2.1. **Bedingungen:** compile-commands.json und find-all-symbols-db.yaml müssen vorhanden sein. Für die Symboldatenbank bei bestehendem compile-commands.json `path/to/llvm/source//`/
`clang-tools-extra/clang-include-fixer/find-all-symbols/tool/run-find-all-symbols.py` aufrufen.

2.2. **Befehl:** `/path/to/clang-include-fixer -db=yaml path/to/file/with/missing/include.cpp`

3. ClangCheck

Dieses Tool hilft mit dem AST von clang zu arbeiten.

3.1. **Bedingungen:** Es sollte ein Verständnis des AST vorhanden sein. Dieses Tool ist kein Einfaches. `-ast-print` - Erstellt ASTs und gibt sie dann aus. `-ast-dump` - Erzeugt ASTs und gibt sie dann als Debug-Dump aus. `-ast-dump-filter=<string>` - Mit -ast-dump oder -ast-print werden nur AST-Deklarationsknoten ausgegeben, die eine bestimmte Teilzeichenkette in einem qualifizierten Namen haben. Verwenden Sie -ast-list, um alle filterbaren Deklarationsknotennamen aufzulisten. `-ast-list` - Erstellt ASTs und gibt die Liste der qualifizierten Namen der Deklarationsknoten aus.

Received by the editor Received by the editor.

3.2. **Befehl:** `clang-check source.cpp`

4. Clang-Tidy

Dieses Tool kann statische Regeln, die die Codequalität betreffen, überprüfen.

4.1. **Bedingungen:** compile-commands.json und .clang-tidy müssen vorhanden sein. In .clang-tidy kann z.B. `Checks: '-*,cppcoreguidelines-*'` stehen. Für eine komplette Liste https://clang.llvm.org/extra/clang-tidy/index.html betrachten.

4.2. **Befehl:** `clang-tidy test.cpp --config= --verify-config`

5. Modularize

Dieses Tool überprüft ob Headerfiles zusammen konsistent funktionieren.

5.1. **Bedingungen:** Ein .txt File,welches die zu überprüfenden Header spezifiziert, muss vorhanden sein. Hier bei gilt:
- ein Headerfile pro Zeile.
- `header1.h: header2.h` bedeutet, dass header1.h von header2.h abhängt.

5.2. **Befehl:** `modularize files.txt`

6. pp-trace

Dieses Tool verfolgt Präprozessoraktivitäten.

6.1. **Bedingungen:** Kenntnis über die verwendeten Präprozessorbefehle. Für eine komplette Liste über die möglichen Tests https://clang.llvm.org/extra/clang-tidy/index.html betrachten.

6.2. **Befehl:** `pp-trace -callbacks FileChanged source.cpp`

7. Clang-Rename

Dieses Tool kann die im Code verwendeten Namen ändern.

7.1. **Bedingungen:** Es empfielt sich ein test.yaml File mit den entsprechnden Änderungen anzulegen:

```
---
- QualifiedName:  foo1
  NewName:        bar1
- QualifiedName:  foo2
  NewName:        bar2
...
```

— Aktuell kann leider nur ein File pro Aufruf geändert werden.

7.2. **Befehl:** `clang-rename -input=test.yaml test.cpp`

8. clangd

Dieses Tool stellt einen Language Server bereitet. Genauer braucht man es nur beim Erstellen einer eigenen IDE zu kennen.

9. ClangFormat

Dieses Tool formatiert Code automatisch. Am Besten als IDE Plugin nutzen.

10. Clang-Doc

Erstellt automatische Dokumentationen. Nicht bereit für produktiven Einsatz.

11. LibTooling

Zur Erstellung eigener Tools https://clang.llvm.org/docs/LibTooling.html nutzen.