

# **GCC - die GNU Compiler Collection**

[Jana Jaeger](#)

## **Was ist GCC?**

Zum Übersetzen von Programmen in Maschinsprache wird ein Compiler benötigt. Der Linux-Compiler schlechthin ist der GCC.

Früher nannte sich der GCC noch GNU C Compiler. Diese Bezeichnung ist aber schon lange nicht mehr gerechtfertigt, da GCC nicht mehr nur für das Compilieren von C Programmen zuständig ist, sondern Compiler für C, C++, Fortran77, Objective-C und Java enthält.

Neben den eigentlichen Compilern enthält die Sammlung auch noch die nötigen Bibliotheken. Jede Programmiersprache braucht eine Laufzeitumgebung, um Hardwareeigenschaften zu abstrahieren und dem Programmierer ein weitgehend plattformunabhängiges Entwickeln zu ermöglichen.

Die C Bibliothek (glibc) macht hierbei eine Ausnahme - sie ist ein eigenständiges Paket. Sie ist zu sehr plattformabhängig und auch viel zu umfangreich, um im allgemeinen GCC Paket Platz zu finden.

## **Welche Plattformen werden von GCC unterstützt?**

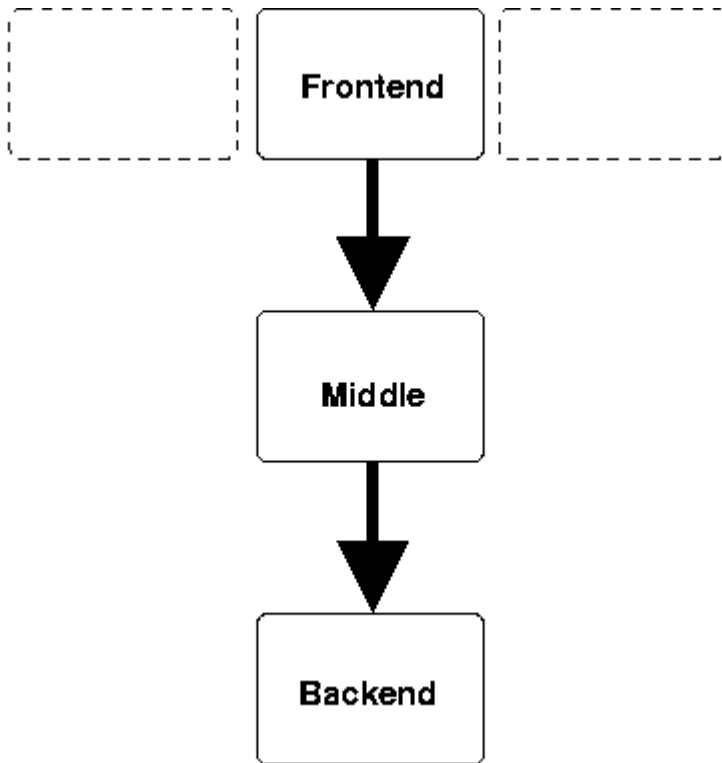
GCC unterstützt eine breite Anzahl von Plattformen und Betriebssystemen. Die unterstützten Betriebssysteme reichen von Linux, BSD, Solaris, HP-UX bis schließlich hin zu Windows.

Die Unterstützung der verschiedensten Architekturen ist noch umfangreicher. Es werden sowohl Embedded Prozessoren, wie Arm, als auch die Standard-CISC und RISC Plattformen wie ix86, Alpha, PowerPC, Sparc und MIPS unterstützt. Mit GCC Version 3.0 wird erstmals ia64 unterstützt, sowie diverse neue Embedded Plattformen.

## **Aufbau des GCC**

Grob gesagt, gliedert sich der GCC in drei Teile: Frontend, Middle und Backend.

- für jede unterstützte Programmiersprache existiert ein separates Frontend. So haben C und C++ zum Beispiel eigene Frontends
- der eigentliche Code wird hier in eine Zwischensprache, RTL oder Register Transfer Language, übersetzt.
- hier werden auf der Zwischensprache Optimierungen des Codes ausgeführt
- der optimierte Code wird in die entsprechende Maschinsprache übersetzt



Bei der Konfiguration der GCC wird festgelegt, für welche Plattform und welche Programmiersprachen Compiler erzeugt werden sollen. Ein Backend erzeugt dabei für genau eine Plattform Code. Ausnahmen sind z.B. die MIPS und SPARC Compiler, die (sofern entsprechend konfiguriert) sowohl 32- als auch 64-Bit Code erzeugen können.

Da es für jede Architektur die entsprechenden Backends und für jede (unterstützte) Programmiersprache das passende Frontend gibt, ist es mit dem GCC sehr einfach möglich, auf einer Plattform (beispielsweise ia32) Programmcode für eine andere Hardware (evtl. noch gar nicht verfügbare, ein Beispiel x86-64) zu compilieren, indem man einfach den entsprechenden Compiler (wichtig das passende Backend ;) installiert und den Code dort durchlaufen lässt. Dieses völlig plattformunabhängige Compilieren nennt man auch **Crosscompilieren**.

## GCC für Anfänger

Nur ein paar kurze Worte zur allgemeinen Anwendung des GCC. Die vorgestellten Optionen sind eher als kleiner Vorgeschmack auf das gedacht, was ernsthafte Programmierer erwartet, die ihre selbstgeschriebenen Werke compilieren wollen, denn als wirkliche Einführung in die Materie.

Um ein kleines C-Programm zu compilieren, braucht es für den Anfang noch nicht viel mehr als diesen Aufruf:

```
gcc dateiname.c
```

Das macht für sich noch wenig Sinn, da das Programm so zwar compiliert wird, aber standardmäßig den Namen `a.out` bekommt. Um also eine ausführbare Datei mit einem vernünftigen Namen zu bekommen, sollte folgender Aufruf genügen:

```
gcc dateiname.c -o dateiname
```

Bis jetzt wird das Programm noch völlig ohne jegliche Optimierung compiliert, mit der Option `-O` und weiteren spezielleren Optionen bekommt der Compiler mitgegeben, wie weit die Optimierungen gehen dürfen.

```
gcc dateiname.c -O<0,1,2 oder 3, und s> -o dateiname
```

liefert optimierten Code. Die Erweiterungen 0 bis 3 (größere Zahlen als 3 sind möglich, 3 ist aber zur Zeit die maximale Optimierungsstufe) sind ein Maß für den Optimierungsgrad. `-O0` ist die Standardeinstellung - es werden dann keine Optimierungen durchgeführt. Bei einem Optimierungsgrad von 3 wird am stärksten optimiert. Je nach Bedarf kann auch die Dateigröße als Optimierungskriterium vorgegeben werden. Mit `-Os` werden nur solche Optimierungen erlaubt, die den Code letztendlich nicht aufblähen.

Weiterhin kann der Programmierer dem Compiler mitgeben, für welchen Prozessortyp der Code optimiert werden soll. Ein Beispiel:

```
gcc dateiname.c -march=i686
```

versucht, speziell für Pentium Pro und seine Nachfolgeprozessoren zu optimieren. Mit Release 3.0 kennt GCC auch Optimierungen für Athlon Prozessoren mit `-march=athlon`. Einen Überblick über den gesamten "Dschungel" der Optionen liefern die Infoseiten zu GCC, per `info gcc` oder im *Konqueror* mit `info: /gcc` aufzurufen.

## GCC und Linux

GCC ist mit Sicherheit nicht der einzige Compiler, der zum Compilieren eines Linux Programmes verwendbar ist. Aber er ist von Beginn an fest mit der Linuxgeschichte verbunden. GCC existierte schon bevor Linus Torvalds auf die Idee mit einem eigenen Betriebssystem kam - und GCC ist so frei, wie es ein Programm von der Lizenz her überhaupt nur sein kann - nämlich unter der GPL. Das heißt, daß alle Änderungen, ob sie nun Betriebssysteme oder neue Architekturen betreffen, schnell und unkompliziert in den Quellcode integriert werden können. Diesen Vorteil haben die anderen (mehr oder weniger frei) verfügbaren Compiler nicht. Zugespitzt kann man sagen, daß GCC **der** Compiler unter Linux überhaupt ist.

## Informationsquellen

- Die Homepage des [GCC-Projekts](#)
- Die [Online-Dokumentation](#) zu GCC 