# Basisbefehle – GNU-Debugger - gdb

Hinweise:

- In allen unten aufgeführten Befehlen steht das Doppelkreuz (#) immer für eine Zahlenangabe.
- Abkürzungen der Befehle sind **grau** unterlegt.

C-, C++ bzw. GNU-Assembler-Programme wie gewohnt, aber mit der Zusatz-Option **-g** kompilieren. Die Option **-g** bewirkt, dass z.B. die Namen aller Variablen und Funktionen noch so erscheinen wie sie im Programm verwendet wurden, d.h. die Symbol-Tabelle wird geladen.

Für C- und C++-Programme:

```
$ gcc -g filename.c
$ g++ -g filename.cpp
```

Für GNU-Assembler-Programme:

```
$ gcc -g filename.s -o filename -no-pie
$ as  -g filename.s -o filename.o && ld filename.o -o filename
```

Debugger starten:

$ **gdb filename**

Es erscheint der gdb-Prompt **(gdb)** in Klammern.

Pfeiltasten ↑ und ↓ :

Mit den Pfeiltasten ↑ und ↓ kann man zu vorherigen Befehlen wandern und sie erneut ausführen, solange man sich nicht im TUI-Modus befindet (s.u.).

Hilfe zu einzelnen Befehlen innerhalb des Debuggers aufrufen:

(gdb) **help** Befehlsname

Gibt Beschreibung eines Befehls aus (z.B. **help print**).

(gdb) **<return>**

Wiederholt den letzten Befehl.

Debugger verlassen:

(gdb) **quit** (evtl. noch mit **y** bestätigen)

Vollständige Dokumentation zur aktuellen gdb-Version:
https://sourceware.org/gdb/download/onlinedocs/gdb.html/index.html

Mit einem Debugger kann man …

- ein Programm Zeile für Zeile durchwandern
- in Funktionen hineinspringen und sie Zeile für Zeile inspizieren (step into)
- zur nächsten Programmzeile springen, ohne dabei eine evtl. vorhandene Funktion zu betreten (step over)
- Breakpoints setzen und das Programm weiterlaufen lassen bis der nächste Breakpoint erreicht ist
- Werte von Variablen und Stackframes inspizieren
- watches, conditional breakpoints setzen, u.v.m.

**step** - Step into

In die Funktion hinein springen, die in der aktuellen Zeile angezeigt wird. Mit **where** kann man sich anzeigen lassen von welcher Funktion die aktuelle aufgerufen wurde (function call stack).

**next** - Step over

Springt zur nächsten Zeile, auch wenn sich in der aktuellen Zeile ein Funktionsaufruf befindet.

**finish** - Step out

Aus der aktuellen Funktion (stack frame) heraus springen.

Den Status des Programms untersuchen:

**where / backtrace**

Gibt den aktuellen Stackframe aus (welche Funktion hat welche andere aufgerufen). Der jeweilige Funktionsname und dessen Argumente werden zuerst ausgegeben.
Die Anzeige **filename.c##** gibt die aktuelle Zeilennummer in der Datei an.

**list [start_line] [, end_line]**

**list** ohne Argumente listet die aktuellen 10 Zeilen auf, in denen man sich gerade befindet. Optional kann ein anderer Start- oder Start- und Endzeile gewählt werden.

**li**

Listet die nächsten 10 Zeilen auf.

**info locals** bzw. **info args**

Gibt die lokalen Variablen der aktuellen Funktion bzw. deren Argumente aus.

**print** name

Kann u.a. folgende Informationen ausgeben:

- den Wert einer Variablen (z.B. **print a**)
- zu einem Zeichen den entsprechenden ASCII-Wert (z.B. **print 'h'**)
- das Ergebnis eines Ausdrucks (z.B. **print a < b**)
- den Inhalt eines Registers (z.B. **print $rax** bzw. **print $xmm0**)
  Bei Ausgabe von Registern wird das %- durch das $-Zeichen ersetzt.
- die aktuell gesetzten Flags (**print $eflags**)
- die Adresse eines Labels (z.B. **print &labelname**)
- u.v.m.

**print**-Anweisungen werden durchgezählt und man kann sie mit **print $#** wiederholt ausführen.

**info** argument

mögliche Argumente:

- **frame** Infos über den aktuellen Stackframe einschl. seiner Adresse, der Adresse des vorherigen Stackframes, Inhalte spezieller Register (z.B. rip), Funktionsargumente und lokale Variablen.
- **stack** agiert wie **where** oder **backtrace**
- **registers** Listet den Inhalt aller Register auf
- **functions** Listet die Signatur aller Funktionen auf (sehr viele!)
- **address labelname**
  Gibt die Adresse eines Labels namens 'labelname' aus

  u.s.w.

Inhalt von Adressen ausgeben:

**x [/Nuf] expr** **x =** e**x**amine memory

N: Anzahl auszugebender Einheiten

u: Größe der Einheiten
(nicht zu verwechseln mit Assembler-Suffixes b, w, l, q)
**b**: einzelne Bytes
**h**: halfwords (2 Bytes)
**w**: words (4 Bytes)
**g**: giant words (8 Bytes)

f: Ausgabeformat
**x**: hexadezimal
**d**: dezimal (signed)
**u**: dezimal (unsigned)
**t**: binär
**a**: Adresse (absolut / relativ)
**c**: Zeichen
**f**: Fließkommazahl

**x/s expr** Gibt Zeichenkette ab Adresse (expr) aus

**x/i expr** Gibt Maschinenbefehl an Adresse (expr) aus

An ausgewählten Positionen anhalten:

**start**

Bewirkt, dass das Programm bis vor der ersten "bedeutsamen" Anweisung (temporärer Breakpoint) ausgeführt wird (z.B. bis **int main()**). Funktioniert nur, wenn der Einsprungpunkt "main" vorhanden ist.

**break [line#]**

Setzt in Zeile# einen Breakpoint (z.B. **break 20**). Mit **break [file:line#]** kann man auch Breakpoints in anderen Dateien setzen, die zum aktuellen Projekt gehören.

**break [line#] if [expression]**

Pausiert in Zeile#, wenn ein bestimmter Ausdruck wahr ist (conditional breakpoint) (z.B: **break 20 if a == b**).
Nützlich z.B. in Schleifen oder bei wiederholt aufgerufenen Funktionen bis ein bestimmter Ausdruck wahr ist.

**break functionname**

Pausiert beim Beginn der Funktionsdefinition (z.B.: **break main**).

**info break**

Listet alle gesetzten Breakpoints mit ihren zugehörigen Nummern und Zeilennummern auf.

**continue**

Läuft weiter bis zum nächsten Breakpoint.

**run**

Alternative zu **start**, wenn bereits ein Breakpoint gesetzt wurde.

**run arg1 [, arg2, …]**

Startet das Programm mit Argumenten.

**clear [line#]**

Löscht Breakpoint in bestimmter Zeile.

**delete [breakpoint#]**

Löscht Breakpoint mit einer bestimmten Nummer.

**delete**

Löscht alle Breakpoints.

**Watchpoints** pausieren, wenn der Wert einer ausgewählten Variablen sich ändert. Es wird dann immer der alte und der neue Wert ausgegeben. Der **watch**-Befehl funktioniert nur in dem Block, in dem die Variable gültig ist.

**watch variable name 1 [, variable name 2, …]**

z.B.: **watch a** → **continue** → Ausgabe:

```
Hardware watchpoint #: variable name(s)
Old value = 30
New value = 24
main () at filename.c:line#
```

**watch expression**

z.B.: **watch a > 5** hält an, wenn a > 5 erreicht wurde
Besonders nützlich in Schleifen

**disassemble**

Listet den Assembler-Code (Object dump) des Programms auf.

**disassemble functionname**

Listet nur den Assembler-Code einer bestimmten Funktion auf.

Statusflags setzen bzw. zurücksetzen

Positionen der wichtigsten Flags im **Eflags**-Register:

 0 – **CF** (Carry flag)
 1 – **PF** (Parity flag)
 6 – **ZF** (Zero flag)
 7 – **SF** (Sign flag)
11 – **OF** (Overflow flag)

**set $eflags |= (1 << 6)**

Setzt das Zeroflag.

**set &eflags &= ~(1 << 6)**

Setzt das Zeroflag zurück.

Hinweis:

Das Systemflag IF (Interrupt Enable Flag) ist immer gesetzt und sollte auch nicht zurückgesetzt werden.

Registerinhalt verändern:

**set $rax = 25000**

Ändert den Inhalt des Registers **rax**.

**set $xmm0 = 1.8f**

Verändert den Inhalt des Registers **xmm0** in den **float**-Wert 1.8.

**set $xmm1 = 1.8**

Verändert den Inhalt des Registers **xmm1** in den **double**-Wert 1.8.

**TUI** - Text-User-Interface Modus

**layout** ist eine Terminalschnittstelle, die es dem Benutzer ermöglicht, die Quelldatei während des Debuggens anzuzeigen. Der TUI-Modus ist standardmäßig aktiviert, wenn man **gdb** als **gdb tui** aufruft.

TUI-Modus ein- und ausschalten:

**tui enable or <strg> + x + a**

Schaltet den TUI-Modus ein.

**tui disable bzw. <strg> + x + a**

Schaltet den TUI-Modus wieder aus.

**<strg>x 2**

Öffnet mehrere Fenster.

**tui reg vector**

Zeigt größere Register wie **xmm#** bzw. **ymm#** an.

**<strg>l**

Erneuert die Ausgabe der Fenster (Refresh)

Befehle in Verbindung mit **layout**:

Der jeweilige Parameter steuert, welche zusätzlichen Fenster angezeigt werden:

**layout next**

Zeigt das nächste Layout an.

**layout prev**

Zeigt das vorherige Layout an.

**layout src**

Zeigt das Quell- und Befehlsfenster an.

**layout asm**

Zeigt das Assembler- und Befehlsfenster an.

**layout split**

Zeigt die Quell-, Assembler- und Befehlsfenster an.

**layout regs**

Zeigt im src-Layout das Register-, Quell- und Befehlsfenster an.
Wenn man sich im split- oder asm-Modus befindet, werden die Register-, Assembler- und Befehlsfenster angezeigt.

Mit **focus** zwischen Fenstern wechseln:

**focus next**

Aktiviert das nächste Fenster zum Scrollen.

**focus prev**

Aktiviert das vorherige Fenster zum Scrollen.

**focus src**

Aktiviert das Quellfenster zum Scrollen.

**focus asm**

Aktiviert das Assembler-Fenster zum Scrollen.

**focus regs**

Aktiviert das Registerfenster zum Scrollen.