GDB GNU debugger

START HERE

`gdb --args <prog> <args>`
Debug a program with arguments

`backtrace`
Show the call stack

`break <func>`
Stop at a function

`print <expr>`
Evaluate and show a value

`run`
Start the program

`continue`
Resume until the next stop

`next / step`
Over / into the next line

`layout src`
Source window (TUI)

Break break, watch → `run` → **Stopped** frame, locals → `step` / `continue` → **Inspect** print, x, info → `repeat` → **Resume** next, finish, continue

CONCEPTS

Compile with -g (and -O0 for sane stepping), otherwise GDB shows no source lines or local variables.

Press Enter to repeat the last command, and use short forms: b, r, c, n, s, bt, p, i.

START AND ATTACH

`gdb <prog>`
Load a program

`gdb --args <prog> <args>`
Load a program with arguments

`gdb -p <pid>`
Attach to a running process
Often needs root or ptrace permission (kernel.yama.ptrace_scope)

`gdb <prog> <core>`
Inspect a core dump

`gdb -q -batch -ex 'bt' <prog> <core>`
Non-interactive: print a backtrace and exit

`run [args]`
Start the program

`start`
Run and stop at the start of main

`set args <args>`
Set arguments for later runs

`run < <file>`
Redirect input as in a shell

`attach <pid>`
Attach from inside GDB

`detach`
Release the process and keep it running

`kill`
Kill the program being debugged

`quit`
Leave GDB

`set follow-fork-mode child`
Follow the child after fork

BREAKPOINTS

`break <func>`
Break at a function

`break <file>:<line>`
Break at a line of a file

`break <line>`
Break at a line of the current file

`break <loc> if <cond>`
Break only when the condition holds

BREAKPOINTS (cont.)

`tbreak <loc>`
One-shot breakpoint, deleted after first stop

`rbreak <regex>`
Break on all functions matching the regex

`info breakpoints`
List breakpoints and watchpoints

`delete [<n>]`
Delete one breakpoint, or all

`disable <n> / enable <n>`
Turn a breakpoint off / on

`condition <n> <expr>`
Set or change a breakpoint condition

`ignore <n> <count>`
Skip the next hits of a breakpoint

`commands <n>`
Attach commands run when it hits (end with 'end')

WATCHPOINTS

`watch <var>`
Stop when a value changes
A watch on a local ends when its frame is left

`watch -l <expr>`
Watch the memory address, not the scope

`rwatch <var>`
Stop when a value is read

`awatch <var>`
Stop on read or write

RUN CONTROL

`continue [<n>]`
Resume; optionally skip n-1 further hits here

`next [<n>]`
Step over calls to the next line

`step [<n>]`
Step into functions with debug info

`finish`
Run until the current function returns
Prints the returned value

`until`
Finish the loop without stepping through it

`until <loc>`
Run to a location in this frame

`stepi / nexti`
One machine instruction

RUN CONTROL (cont.)

`call <func>(<args>)`
Call a function in the program

`Ctrl+C`
Interrupt the running program

STACK

`backtrace`
Show the call stack

`backtrace <n> / backtrace -<n>`
Innermost n frames / outermost n frames

`backtrace full`
Include local variables in each frame

`thread apply all bt`
Backtrace of every thread

`frame <n>`
Select a frame

`up [<n>] / down [<n>]`
Move toward the caller / callee

`info locals`
Local variables of the frame

`info args`
Arguments of the frame

INSPECT DATA

`print <expr>`
Evaluate and show an expression

`print/x <expr>`
Print in hex (also d, o, t, c, a)

`print *<ptr>`
Dereference a pointer

`print <arr>@<n>`
Show n elements from the first one

`set var <var> = <val>`
Change a variable

`display <expr>`
Show an expression at every stop

`x/<n><f><u> <addr>`
Examine memory
Format x d s i c; unit b h w g, e.g. x/8xw $sp or x/s <str>

`x/<n>i $pc`
Disassemble n instructions at the PC

`ptype <expr-or-type>`
Show the full type definition

`whatis <expr>`
Show the type of an expression

`info registers`
Show general-purpose registers

INSPECT DATA (cont.)

`set print pretty on`
Print structs one field per line

SOURCE AND SYMBOLS

`list`
Show source around the current line

`list <func>`
Show source around a function

`info functions <regex>`
Search function names

`info threads`
List threads

`thread <n>`
Switch to a thread

TEXT UI

`gdb -tui <prog>`
Start with the text UI

`layout src`
Source window above the command line

`layout split`
Source and disassembly

`focus cmd`
Give the command window the keys (also src, asm)

`Ctrl+X A`
Toggle the text UI on and off

`Ctrl+L`
Redraw a garbled screen

RECIPES

Crash post-mortem

```
ulimit -c unlimited
./<prog>            # crashes, writes core
gdb <prog> <core>
(gdb) bt full
```

Stop in a loop on a condition

```
(gdb) break <file>:<line> if i == 99
(gdb) run
(gdb) display total
(gdb) next
```

Who overwrites this variable?

```
(gdb) break main
(gdb) run
(gdb) watch -l <var>
(gdb) continue    # stops at the writing line
(gdb) bt
```

Log and go on

```
(gdb) break <func>
(gdb) commands
> silent
> printf "arg=%d\n", <arg>
> continue
> end
```