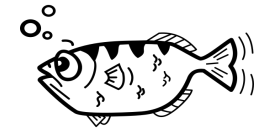


GDB Cheat Sheet



RUNNING

<code>gcc -g -o program program.c</code>	Compile your code with debugging information
<code>gcc -On -o program program.c</code>	Compile code using Optimization On <code><n:0-3></code> : Be very careful if you use On (<code>n>2</code>) you will have difficulties in debugging code.
<code>gdb <program> [core dump]</code>	Start GDB (with optional core dump).
<code>gdb --args <program> <args></code>	Start GDB and pass arguments
<code>gdb --pid <pid></code>	Start GDB and attach to the process.
<code>set args <args...></code>	Set arguments to pass to the program to be debugged.
<code>run <args></code>	Run the program to debug, possible to pass arguments.
<code>quit / q</code>	Quit the debug mode.

BREAKPOINTS

<code>break <function/line/file:line></code>	Set a new breakpoint.
<code>delete <breakpoint#></code>	Delete a breakpoint.
<code>clear</code>	Delete all breakpoints.
<code>enable <breakpoint#></code>	Enable a breakpoint.
<code>disable <breakpoint#></code>	Disable a breakpoint.

WATCHPOINTS

<code>watch <function/line/file:line></code>	Set a new watchpoint.
<code>enable <watchpoint#></code>	Enable watchpoint.
<code>disable <watchpoint#></code>	Disable watchpoint.
<code>delete <watchpoint#></code>	Delete watchpoint.

CONDITIONS

<code>break/watch <function/line/file:line> if <condition></code>	Break/watch at the given location if the condition is met. Conditions can be a C expression that evaluates a true or false.
<code>condition <breakpoint#> <condition></code>	Set/change the condition of an existing break- or watchpoint.

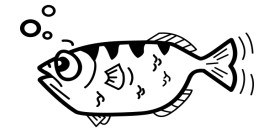
STACK ANALYSIS

<code>backtrace / bt / where</code>	Show call stack.
<code>backtrace full / bt full where full</code>	Show call stack, also print the local variables in each frame.
<code>frame <frame#></code>	Select the stack frame to operate on.
<code>list / l</code>	Display the first 10 lines of program code.
<code>list file.c:n</code>	Display from line (n-5) to (n+5) of the file.c program.
<code>layout src / asm</code>	Display the c/asm program code.
<code>layout split</code>	Display both assembler and source code.
<code>layout reg</code>	Display CPU registers status.

INFORMATIONS

<code>info locals</code>	Display all local variables content.
<code>info registers</code>	Display CPU registers values.
<code>info args</code>	Display all arguments in use.
<code>info threads</code>	Display a summary of all threads currently in your program.
<code>info signals</code>	List all signals and how they are currently handled.

GDB Cheat Sheet



STEPPING

<code>step / s <x: number_of_steps></code>	Go to the x next instructions into function line code.
<code>next / n <x: number_of_steps></code>	Go to the x next instructions over function line code.
<code>finish</code>	Continue until the current function returns.
<code>continue</code>	Continue normal execution until the next watchpoint or breakpoint.

MEMORY ACCESS

<code>x <address or &variable></code>	Examine the memory address value.
<code>x/nts <address or &variable></code>	Examine memory mapping content: <ul style="list-style-type: none">• n: number of memory cells to show.• t: Type of the data to show:<ul style="list-style-type: none">○ <i>a: Pointer.</i>○ <i>c: Read as integer, print as character.</i>○ <i>d Integer, signed decimal.</i>○ <i>f: Floating point number.</i>○ <i>o: Integer, print as octal.</i>○ <i>s: string.</i>○ <i>t: Integer, print as binary.</i>○ <i>u: Integer, unsigned decimal.</i>○ <i>x: Integer, print as hexadecimal.</i>• s: Size of the data to show:<ul style="list-style-type: none">○ <i>b: Byte</i>○ <i>h: Half-word (2 bytes)</i>○ <i>w: Word (4 bytes)</i>○ <i>g: Giant word (8 bytes).</i>

PROCESS & THREAD & SIGNAL

<code>gdb --pid <pid></code>	Start GDB and attach to the process.
<code>attach <pid></code>	Attach GDB to a running process.
<code>detach <pid></code>	Release process from GDB control. Detaching the process continues its execution.

<code>set follow-fork-mode mode</code>	parent: The original process is debugged after a fork. The child process runs unimpeded. child: The new process is debugged after a fork. The parent process runs unimpeded. ask: The debugger will ask for one of the above choices
<code>show follow-fork-mode</code>	Display the current debugger response to a fork or vfork call
<code>Kill</code>	Kill the children's process.
<code>thread thread_nb</code>	Make thread number thread_nb the current thread.
<code>handle <signal> <options></code>	Set how to handle signals. Options are: <ul style="list-style-type: none">• (no)print: (Don't) print a message if signals occur.• (no)stop: (Don't) stop the program if signals occur.• (no)pass: (Don't) pass the signal to the program.

VARIABLES

<code>print <variable name></code>	Display variable content.
<code>print *<array_name>@length</code>	Display arrays values.
<code>print \$register_name</code>	Display a CPU <i>register_name</i> value.
<code>set var <variable_name>=<value></code>	Change the variable content to the given value.
<code>print \$register_name</code>	Display a CPU <i>register_name</i> value.
<code>return <expression></code>	Force the current function to return immediately, passing the given value.