GDB Cheat Sheet

RUNNING

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

BREAKPOINTS

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

WATCHPOINTS

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

CONDITIONS

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

STACK ANALYSIS

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

INFORMATIONS

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

GDB Cheat Sheet

STEPPING

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

MEMORY ACCESS

x <address or &variable>	Examine the memory address value.

x/nts <address or &variable>	Examine memory mapping content:
	• **n: number of memory cells to show.**
	• **t: Type of the data to show:**
	  ○ *a: Pointer.*
	  ○ *c: Read as integer, print as character.*
	  ○ *d Integer, signed decimal.*
	  ○ *f: Floating point number.*
	  ○ *o: Integer, print as octal.*
	  ○ *s: string.*
	  ○ *t: Integer, print as binary.*
	  ○ *u: Integer, unsigned decimal.*
	  ○ *x: Integer, print as hexadecimal.*
	• **s: Size of the data to show:**
	  ○ *b: Byte*
	  ○ *h: Half-word (2 bytes)*
	  ○ *w: Word (4 bytes)*
	  ○ *g: Giant word (8 bytes).*

PROCESS & THREAD & SIGNAL

gdb --pid <pid>	Start GDB and attach to the process.
attach <pid>	Attach GDB to a running process.
detach <pid>	Release process from GDB control. Detaching the process continues its execution.
set follow-fork-mode mode	***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
show follow-fork-mode	Display the current debugger response to a fork or vfork call
Kill	Kill the children's process.
thread thread_nb	Make thread number ***thread_nb*** the current thread.
handle <signal> <options>	Set how to handle signals. Options are: • (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

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