one-page gdb cheat sheet v1.0

1. start gdb

// simply start gdb
gdb <program>

// use if your program has arguments
gdb --args <program> <args>

// use a file with gdb commands
gdb -x <gdb_file> <program>

// directly start debugging (skips step 2)
gdb --ex=r <program>

// attach to process by pid
gdb --pid <pid>

2. gdb started, pre-debug

// execute commands from <file>
source <file>

2.1 breakpoints

// break execution at <where>
// <where> can be a line number, a function, etc.
break <where> or **b <where>**

// break at line number 42 in current source file
break 42

// break at line number 42 in source file foobar.c
break foobar.c:42

// break when calling function doCalc
break doCalc

// there are also conditional breakpoints;
// break at function doCalc if x > 0
break doCalc if x>0

// show infos for all bps; optionally only for bp n
info breakpoints [*n*]

// delete all bps; optionally only bp n
delete [*n*]

// disable all bps; optionally only bp n
disable [*n*]

// opposite of disable
enable [*n*]

// save breakpoints to file
save breakpoints <file>

2.1 watchpoints

// program is stopped if <what> is written to
watch <what>

// stopped if read
rwatch <what>

// stopped in both cases
awatch <what>

// also addresses can be watched with *
watch *<address>

// and registers as well with $
watch $<register>

// The following commands are similar to breakpoints:
info watchpoints

3. while debugging

// simply start the debugging
run or **r**

3.1 visualization

// for fancy views: gdb TUI (text-user-interface)
// show {assembly code, source code, regs}
layout {asm,src,regs}

// show both source and assembly code
layout split

// change window focus in tui mode
ctrl + x

// close all tui windows
tui disable

3.2 printing

// print the value of what
print <what>

// strings are usually cutoff after 200 chars
// use this to print unlimited chars
set print elements 0

// print all local variables
info locals

// print all function arguments
info args

3.2 stepping

// step to next instruction; go into function
step [*n*] or **s [*n*]**

// step to next instruction; don't go intro function
next [*n*] or **n [*n*]**

// similar to step but with machine instructions
stepi [*n*] or **si [*n*]**

// similar to next but with machine instructions
nexti [*n*] or **ni [*n*]**

// step out of function
finish or **fin**

// continue execution
continue or **c**

3.3 backtrace

// show current call stack; optionally with local vars
backtrace [full] or **bt [full]**

// select frame n
frame *n*

3.4 manipulation

// set variable or address to value
set var {<variable>,<address>} = <value>

// directly returns from the function
return <expression>

4. end gdb

quit or **q** // end gdb

{a,b}	choose either a or b
ctrl	hit key "ctrl"
n	replace by an integer number
<term>	use your brain to replace this term

delete [*n*]
disable [*n*]
enable [*n*]

popular stack overflow question

[] anything between the brackets is optionally

created by chciken with ❤️