

CSCI 2330 GDB Reference Sheet

Start

<code>gdb myprog</code>	Launch myprog in gdb (basic mode)
<code>gdb -tui myprog</code>	Launch myprog in gdb (fancy mode)
<code>layout src</code>	Show source code while debugging
<code>layout asm</code>	Show assembly code while debugging
<code>Control-L</code>	Refresh layout (e.g., after program output)

Run and Stop

<code>help [h]</code>	Get information about gdb
<code>quit [q]</code>	Exit gdb
<code>run [r]</code>	Run program
<code>run 1 2 3</code>	Run with command-line arguments 1 2 3
<code>run < in.txt</code>	Run with input redirected from in.txt
<code>kill [k]</code>	Stop program
<code>Control-D</code>	Exit gdb
<code>Control-C</code>	Stop the currently running gdb command
<code>make</code>	Run make to rebuild without leaving gdb

Breakpoints

<code>break [b]</code>	Set breakpoint at current location
<code>break sum</code>	Set breakpoint at entry to function sum
<code>break 20</code>	Set breakpoint at line 20 in current file
<code>break prog.c:20</code>	Set breakpoint at line 20 in prog.c
<code>break *0x80483c3</code>	Set breakpoint at address 0x80483c3
<code>delete [d]</code>	Delete all breakpoints
<code>delete 1</code>	Delete breakpoint #1 (from "info break")
<code>disable 1</code>	Disable breakpoint #1
<code>enable 1</code>	Enable breakpoint #1
<code>clear sum</code>	Clear breakpoints at entry to function sum

Execute

<code>step [s]</code>	Execute one C line
<code>next [n]</code>	Execute one C line (treats functions as one line)
<code>stepi [si]</code>	Execute one ASM instruction
<code>stepi 4</code>	Execute four ASM instructions
<code>nexti [ni]</code>	Execute one ASM instruction (treats function as one instruction)
<code>continue [c]</code>	Execute until next breakpoint
<code>until 3</code>	Execute until breakpoint #3
<code>finish</code>	Execute until current function returns
<code>call sum(1, 2)</code>	Call sum(1, 2) and print return value

Context

<code>backtrace [bt]</code>	Print current address & stack backtrace
<code>info [i]</code>	Print info about program state (see below)
<code>info program</code>	Print current status of the program
<code>info break</code>	Print status of breakpoints
<code>info frame</code>	Print info about current stack frame
<code>info register</code>	Print registers and their contents

Examine Code

<code>disas</code>	Disassemble current function
<code>disas sum</code>	Disassemble function sum
<code>disas 0x80483b7</code>	Disassemble function around 0x80483b7
<code>disas 0x80483b7 0x80483c7</code>	Disassemble within address range
<code>print /x \$rip</code>	Print program counter in hex
<code>print /d \$rip</code>	Print program counter in decimal
<code>print /t \$rip</code>	Print program counter in binary

Examine Data

<code>print [p]</code>	Print expression (last value by default)
<code>print foo</code>	Print value of foo
<code>print /x foo+5</code>	Print value of (foo+5) in hex
<code>print /d 0xAB</code>	Print 0xAB in decimal
<code>print /d \$rax</code>	Print contents of register %rax in decimal
<code>print /x \$rax</code>	Print contents of register %rax in hex

<code>x/FMT ADDRESS</code>	Examine memory at ADDRESS using format FMT
<code>x/g 0xbffff890</code>	Examine 8-byte word at address 0xbffff890
<code>x/g \$rsp</code>	Examine 8-byte word at address \$rsp
<code>x/w \$rsp</code>	Examine 4-byte word at address \$rsp
<code>x/wd \$rsp</code>	Examine 4-byte word at address \$rsp in decimal
<code>x/2w \$rsp</code>	Examine two 4-byte words at address \$rsp
<code>x/2wd \$rsp</code>	Examine two 4-byte words at address \$rsp in decimal
<code>x/s 0xbffff890</code>	Examine string stored at 0xbffff890
<code>x/6bc \$rsp</code>	Examine six bytes at address \$rsp as chars
<code>x/10i sum</code>	Examine first 10 instructions of func sum
<code>x/20b sum</code>	Examine first 20 opcode bytes of func sum

<code>display /FMT EXPR</code>	Print expression EXPR using format FMT each time execution stops
<code>display</code>	Show current auto-display expressions
<code>undisplay NUM</code>	Remove expression NUM from auto-display

Formats: x/[NUM][SIZE][FORMAT]

If not given, uses sensible default or last-used format

NUM = number of objects to display

SIZE = size of each object

b = 1 byte

h = 2 bytes ("half word")

w = 4 bytes ("word")

g = 8 bytes ("giant/quad word")

FORMAT = format for displaying each object

d = decimal

x = hexadecimal

t = binary

a = address (pointer)

c = character

s = string