

GDB - Gnu Debugger - Initiation

<code>gdb -q ./<file></code>	Start GDB in quiet mode
<code>gdb -p <pid></code>	Attach to process-id
<code>gdb -c <core> ./<file></code>	Load up a core file and the program

Those commands are executed to start GDB.

GDB - Commands - Run a program

<code>run</code>	<code>r</code>	Start the program
<code>run</code>	<code>r</code>	Start with an argument
<code>testarg</code>	<code>testarg</code>	

GDB - Commands - Registers

<code>info registers</code>	<code>i r</code>	Show default registers
<code>info registers</code>	<code>i r a</code>	Show all registers
<code>info registers</code>	<code>i r</code>	Show EAX register
<code>eax</code>	<code>eax</code>	

Commands for showing the content of registers.

GDB - Commands - Examine

<code>x \$eax</code>	Examine address in EAX
<code>x/i \$esp</code>	Examine address at ESP interpret as instruction
<code>x/s 0xffffffff</code>	Examine address interpret as string
<code>x/4s 0xffffffff</code>	Print from that address 4 times
<code>x/4xb</code>	Examine in HEX repeat 4 times show in Bytes
<code>disassemble /</code>	Disassemble at current position
<code>disas _start</code>	Disassemble from label _start

GDB - Commands - Examine (cont)

<code>print / p</code>	Print address of libc system
------------------------	------------------------------

Note: Examine needs valid addresses to function. Unit sizes: b, Bytes; h, Halfwords (two bytes); w, Words (four bytes); g, Giant words (eight bytes).

GDB - Commands - Breakpoint

<code>break _start</code>	<code>b _start</code>	Set a breakpoint at the label _start
<code>break 5</code>	<code>b 5</code>	Breakpoint at source line 5
<code>break *0x443-32211</code>	<code>b *0x443-32211</code>	Breakpoint at address/offset

GDB - Commands - Stepping

<code>step</code>	<code>s</code>	Step per line of source.
<code>stepi</code>	<code>si</code>	Step per machine instruction
<code>continue</code>	<code>c</code>	Continue program execution

GDB - Commands - Set and Call

<code>call (int) mprotect(0xDEADBEEF, 0x1000, 1)</code>	Execute mprotect() in debuggee context.
<code>call strcpy(0xdeadbeef, "hacky")</code>	Write hacky to addr 0xdeadbeef
<code>set follow-fork-mode child</code>	Follow newly created childs
<code>set (char [SIZE]) 0xdeadbeef = "my_new_array"</code>	Write data to address
<code>set {int}0xdeadbeef = 4</code>	Set value at address to 4
<code>set \$eax = 0xdeadbeef</code>	Set value of register EAX to 0xdeadbeef

GDB-GEF - Overview

<code>gdb-gef</code>	Start gdb-gef at commandline
<code>gef help</code>	Show help of GEF
<code>start</code>	Start program with auto breakpoints set
<code>kill</code>	Kill current process
<code>context</code>	<code>ctx</code> Show context
<code>checksec</code>	Check security features
<code>vmmap</code>	Show virtual memory map
<code>python-in-teractive</code>	<code>pi</code> Start Python Interpreter
<code>python-in-teractive</code>	<code>pi 23*5</code> Use python interpreter and calculate 23*5

GDB-GEF - Configuration

<code>gef config</code>	Show running configuration
<code>gef config context</code>	Configure GEF context
<code>gef config context.show_opcode_size 8</code>	Set the opcode output to length of 8
<code>gef config context.layout "legend regs stack memory"</code>	Set only for widgets as output
<code>gef save</code>	Save running configuration

Extra configurations for GDB-GEF

GCC - Overview

<code>gcc -m32 <input> -o <output></code>	Compile source for x86_32 arch.
<code>gcc -m32 <input> -o <output> -z execstack</code>	Compile with executable stack
<code>gcc -m32 <input> -o <output> -g</code>	Compile with debug symbols



By therealdash

cheatography.com/therealdash/

Not published yet.

Last updated 16th November, 2023.

Page 1 of 2.

Sponsored by [ApolloPad.com](https://apollopadd.com)

Everyone has a novel in them. Finish

Yours!

<https://apollopadd.com>

NASM - Overview

<code>nasm -f elf32 <input> -o <output>.o</code>	Creates x86_32 object file from assembly.
<code>ld -m elf_i386 <input>.o -o <output></code>	Create x86_32 ELF from object file

OBJDUMP - Overview

<code>objdump -d -M intel <file></code>	Dump the opcodes in Intel Syntax
<code>objdump -s -j <section> <file></code>	Dump only named section

STRACE - Overview

<code>strace <filename></code>	Starts program and tracing it
<code>strace -p <pid></code>	Attaches at process-id
<code>strace -o log.txt <filename></code>	Writes output into a logfile
<code>strace -f <filename></code>	Also log child processes

PWNtools

<code>pwn asm nop</code>	Write NOP opcode
<code>pwn asm nop 'mov eax, 1'</code>	Write NOP and MOV opcode
<code>pwn asm -f string nop</code>	Outputs in \x Notation
<code>pwn disasm 909090</code>	Output the disassembly of three NOPs

PERL - Basics for exploits

<code>perl -e '{print "A"x"1-024"}'</code>	Print 1024 times A
--	--------------------

Student Files

<code>lessons/</code>	Assembler files, aimed at teaching x86_32 basics
<code>shellcode/</code>	Collection of bad shellcodes, students have to improve
<code>skeletons/</code>	Skeleton Code files

Student Files (cont)

<code>exploits/</code>	Exploits shellcode is ran against
<code>tools/</code>	Support tools for the training



By **therealdash**

cheatography.com/therealdash/

Not published yet.

Last updated 16th November, 2023.

Page 2 of 2.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopadd.com>