GDB commands

This article provides information on some basic GDB commands and explains how to use them.

You can also refer to the "Debugging with GDB" web page ([1] from gnu.org).

Contents

1 target remote *host:port* command

This command establishes a TCP connection between the host and a remote target board. The host can be either a host name or a numeric IP address; the port must be a decimal number:

```
(gdb) target remote <IP_Addr_of_Board>:<port>
```

2 info target / info files commands

These two commands are synonymous. They both display the current target information, including the names of the executable and core dump files currently in use by the GDB, and the files from which the symbols were loaded.

The command **help target** lists all the possible targets rather than the current ones.

```
(gdb) info target
```

```
Symbols from "target:/usr/local/bin/hello_world_example".
Remote serial target in gdb-specific protocol:
Debugging a target over a serial line.
       While running this, the GDB does not access memory from...
Local exec file:
       `target:/usr/local/bin/hello_world_example', file type elf32-littlearm.
       Entry point: 0x10434
       0x00010154 - 0x0001016d is .interp
       0x00010170 - 0x00010190 is .note.ABI-tag
       0x00010190 - 0x000101b4 is .note.gnu.build-id
       0x000101b4 - 0x000101f4 is .gnu.hash
       0x000101f4 - 0x00010284 is .dynsym
       0x00010284 - 0x000102e0 is .dynstr
       0x000102e0 - 0x000102f2 is .gnu.version
       0x000102f4 - 0x00010314 is .gnu.version_r
       0x00010314 - 0x00010324 is .rel.dyn
       0x00010324 - 0x0001035c is .rel.plt
       0x0001035c - 0x00010368 is .init
       0x00010368 - 0x000103d0 is .plt
       0x000103d0 - 0x0001058c is .text
       0x0001058c - 0x00010594 is .fini
       0x00010594 - 0x00010610 is .rodata
       0x00010610 - 0x00010618 is .ARM.exidx
       0x00010618 - 0x0001061c is .eh_frame
       0x00020f10 - 0x00020f14 is .init_array
       0x00020f14 - 0x00020f18 is .fini_array
       0x00020f18 - 0x00021000 is .dynamic
       0x00021000 - 0x0002102c is .got
       0x0002102c - 0x00021034 is .data
       0x00021034 - 0x0002103c is .bss
       0xb6fce0f4 - 0xb6fce1b8 is .hash in target:/lib/ld-linux-armhf.so.3
       0xb6fce1b8 - 0xb6fce298 is .gnu.hash in target:/lib/ld-linux-armhf.so.3
       0xb6fce298 - 0xb6fce478 is .dynsym in target:/lib/ld-linux-armhf.so.3
       0xb6fce478 - 0xb6fce630 is .dynstr in target:/lib/ld-linux-armhf.so.3
       0xb6fce630 - 0xb6fce66c is .gnu.version in target:/lib/ld-linux-armhf.so.3
       0xb6fce66c - 0xb6fce6c8 is .gnu.version_d in target:/lib/ld-linux-armhf.so.3
       0xb6fce6c8 - 0xb6fce808 is .rel.dyn in target:/lib/ld-linux-armhf.so.3
       0xb6fce808 - 0xb6fce840 is .rel.plt in target:/lib/ld-linux-armhf.so.3
       0xb6fce840 - 0xb6fce8a8 is .plt in target:/lib/ld-linux-armhf.so.3
       0xb6fce8c0 - 0xb6fea3dc is .text in target:/lib/ld-linux-armhf.so.3
       0xb6fea3dc - 0xb6fedf74 is .rodata in target:/lib/ld-linux-armhf.so.3
       0xb6fedf74 - 0xb6fedfc8 is .ARM.extab in target:/lib/ld-linux-armhf.so.3
       0xb6fedfc8 - 0xb6fee0b0 is .ARM.exidx in target:/lib/ld-linux-armhf.so.3
       0xb6ffea00 - 0xb6ffeee4 is .data.rel.ro in target:/lib/ld-linux-armhf.so.3
       0xb6ffeee4 - 0xb6ffefac is .dynamic in target:/lib/ld-linux-armhf.so.3
       0xb6ffefac - 0xb6fff000 is .got in target:/lib/ld-linux-armhf.so.3
       0xb6fff000 - 0xb6fff830 is .data in target:/lib/ld-linux-armhf.so.3
       0xb6fff830 - 0xb6fff908 is .bss in target:/lib/ld-linux-armhf.so.3
```

```
        0xb6ffd0d4 - 0xb6ffd0fc is .hash in system-supplied DSO at 0xb6ffd000
        0xb6ffd0fc - 0xb6ffd14c is .dynsym in system-supplied DSO at 0xb6ffd000
        0xb6ffd14c - 0xb6ffd190 is .dynstr in system-supplied DSO at 0xb6ffd000
        0xb6ffd190 - 0xb6ffd19a is .gnu.version in system-supplied DSO at 0xb6ffd000
        0xb6ffd19c - 0xb6ffd1d4 is .gnu.version_d in system-supplied DSO at 0xb6ffd000
---Type <return> to continue, or q <return> to quit---
```

3 Program running commands

(gdb) run - *(abbreviation **r**) Runs the program until a breakpoint is found or an error occurs*

(gdb) continue - *(abbreviation **c**) Continues running the program until the next breakpoint or error*

(gdb) finish - *Runs until the current function is finished*

(gdb) step - *(abbreviation **s**) Executes the next program line*

(gdb) step *N* - *(abbreviation **s N**) Executes the next N lines of the program*

(gdb) stepi - *(abbreviation **si**) Executes one machine instruction, then stops and returns to the debugger*

(gdb) next - *(abbreviation **n**) Similar to **s**, except that it does not step into functions*

(gdb) nexti - *(abbreviation **ni**) Executes one machine instruction. If it is a function call, the command proceeds until the function returns.*

(gdb) until - *(abbreviation **u**) Goes up a level in the stack. This command is used to avoid single stepping through a loop more than once*

(gdb) until *N* - *(abbreviation **u N**) Runs the program until you get N lines in front of the current line*

4 Source code commands

(gdb) list - *(abbreviation* ***l****) Lists* ***listsize*** *more lines after or around the previously listed lines*

(gdb) list *<linenum>* - *(abbreviation* ***l*** *<linenum>) Lists* ***listsize*** *lines around a given* ***line*** *in the current file*

(gdb) list *<filename>:<linenum>* - *(abbreviation* ***l*** *<filename>:<linenum>) Lists* ***listsize*** *lines around a given* ***line*** *in a given* ***file***

(gdb) list *<function>* - *(abbreviation* ***l*** *<function>) Lists* ***listsize*** *lines around the beginning of a given* ***function***

- To see the current *listsize* value (**default value is 10**)

```
(gdb) show listsize
Number of source lines gdb will list by default is 10.
```

- To update the *listsize*

(gdb) set listsize *<count>*

5 Breakpoint commands

The **break** command (*abbreviation **b***) allows adding breakpoints in the program. Current breakpoints can also be listed or deleted.

5.1 break *(abbreviation b)*

The command sets a software breakpoint at a specified location, which can be a function name, a line number, or an instruction address.

```
(gdb) b main - Puts a breakpoint at the beginning of the program
```

```
(gdb) b - Puts a breakpoint at the current line
```

```
(gdb) b <n> - Puts a breakpoint at line n
```

```
(gdb) b +<n> - Puts a breakpoint n lines forward from the current line
(gdb) b -<n> - Puts a breakpoint n lines backward from the current line
```

```
(gdb) b <fn> - Puts a breakpoint at the beginning of the fn function
```

```
(gdb) b <filename>:<linenum> - Puts a breakpoint at a given line number in a given file
defined by its name
```

```
(gdb) b <filename>:<function> - Puts a breakpoint at the entry to a given function in a
file defined by its name
```

```
(gdb) b *<address> - Puts a breakpoint at <address> address
```

For example, when running the program to the breakpoint:

```
(gdb) b 16
(gdb) c
Breakpoint 3, main (argc=1, argv=0xbefffc74) at hello_world_example.c:16
16                printf("\nUser space example: hello world from STMicroelectronics\n");
```

5.2 tbreak *(abbreviation tb)*

Sets a breakpoint enabled only for one stop. Argument parameters are the same as for the **break** command.

5.3 hbreak *(abbreviation hb)*

Sets a hardware-assisted breakpoint. Argument parameters are the same as for the **break** command.

5.4 thbreak *(abbreviation thb)*

Sets a hardware-assisted breakpoint enabled only for one stop. Argument parameters are the same as for the **break** command.

5.5 list breakpoints

```
(gdb) info break - (abbreviation i b)
 Num     Type           Disp Enb Address    What
 1       breakpoint     keep y   0x000103d0 in main at hello_world_example.c:16
```

5.6 enable/disable breakpoints

```
(gdb) enable <n> - Enables breakpoint number n. If n is not present, all breakpoints are
enabled
```

```
(gdb) disable <n> - Disables breakpoint number n. If n is not present, all breakpoints
are disabled
```

5.7 delete breakpoints *(abbreviation d)*

```
(gdb) delete <n> - Deletes breakpoint number n. If n not present, all breakpoints are
deleted
```

6 Backtrace command

The **backtrace** command (*abbreviation **bt***) prints a backtrace of the entire stack: one line per frame for all frames in the stack. You can stop the backtrace at any time by typing the system interrupt character, normally Ctrl+c.

```
(gdb) bt
```

For example:

```
(gdb) bt
#0  0xbf101c48 in delta_perf_end ()
#1  0xbf100d04 in delta_vb2_au_queue () at sources/linux-sti/drivers/media/platform/delta
/delta-v4l2.c:1016
#2  0xc04480d0 in __enqueue_in_driver ()
#3  0xc044a300 in vb2_qbuf ()
#4  0xbf0feea4 in delta_qbuf () at sources/linux-sti/drivers/media/platform/delta/delta-
v4l2.c:668
#5  0xc043f150 in v4l_qbuf ()
#6  0xc043e914 in __video_do_ioctl ()
#7  0xc043e444 in video_usercopy ()
#8  0xc043abc0 in v4l2_ioctl () at sources/linux-sti/drivers/media/v4l2-core/v4l2-dev.c:
351
#9  0xc00ea508 in do_vfs_ioctl ()
#10 0xc00ea708 in sys_ioctl ()
warning: Couldn't find the process executable path.
#11 ret_fast_syscall () at sources/linux-sti/arch/arm/kernel/entry-common.S:34
#12 0xb6cc1780 in ?? ()
#13 0xb39b6318 in ?? ()
#14 0xb39b6318 in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
```

7 info threads command

This command displays a summary of all the threads currently in your program.

The GDB displays for each thread, in this order:

1. The thread number assigned by the GDB
2. The target system thread identifier (systag)
3. The current stack frame summary for this thread

(gdb) info threads - *(abbreviation **i threads**)*

For example:

```
(gdb) i threads
  Id   Target Id         Frame
  71   weston-desktop- (TGID:200) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  70   weston-keyboard (TGID:199) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  69   weston (TGID:198) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  68   openvt (TGID:197) 0xc0028f00 in sys_wait4 () at sources/linux-sti/kernel/exit.c:
1666
  67   weston.sh (TGID:190) 0xc0028f00 in sys_wait4 () at sources/linux-sti/kernel/exit.c:
1666
  66   [kworker/u5:0] (TGID:166) 0xc00412a4 in kthread ()
  65   sh (TGID:152)     0xc02da3b0 in n_tty_read ()
  64   avahi-daemon (TGID:151) 0xc057cc34 in unix_stream_recvmsg () at sources/linux-sti
/net/unix/af_unix.c:1897
  63   agetty (TGID:148) 0xc02da3b0 in n_tty_read ()
  62   [mme_manager] (TGID:147) 0xc05e4a04 in __down_killable () at sources/linux-sti
/kernel/semaphore.c:221
  61   [ics_watchdog] (TGID:143) 0xc05e4ad4 in __down_timeout () at sources/linux-sti
/kernel/semaphore.c:221
  60   [ics_nsrv] (TGID:142) 0xc05e4a04 in __down_killable () at sources/linux-sti/kernel
/semaphore.c:221
  59   [ics_admin] (TGID:141) 0xc05e4a04 in __down_killable () at sources/linux-sti/kernel
/semaphore.c:221
  58   weston.sh (TGID:140) 0xc0028f00 in sys_wait4 () at sources/linux-sti/kernel/exit.c:
1666
  57   systemd-logind (TGID:139) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  56   dbus-daemon (TGID:137) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  55   avahi-daemon (TGID:132) 0xc00eaeb0 in poll_schedule_timeout ()
  54   systemd-udevd (TGID:102) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  53   systemd-journal (TGID:84) 0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  52   [kworker/1:3] (TGID:74) 0xc00412a4 in kthread ()
  51   [deferwq] (TGID:64) 0xc00412a4 in kthread ()
  50   [kworker/1:2] (TGID:63) 0xc00412a4 in kthread ()
  49   [kworker/u4:3] (TGID:62) 0xc00412a4 in kthread ()
  48   [irq/247-st-lpm] (TGID:61) 0xc00412a4 in kthread ()
  47   [irq/55-st_therm] (TGID:60) 0xc00412a4 in kthread ()
  46   [rc0] (TGID:59)   0xc00412a4 in kthread ()
  45   [irq/239-fe54100] (TGID:56) 0xc00412a4 in kthread ()
  44   [irq/220-fed4100] (TGID:53) 0xc00412a4 in kthread ()
```

```
  43   [irq/219-fed4000] (TGID:50) 0xc00412a4 in kthread ()
  42   [ftk_touch_wq] (TGID:49) 0xc00412a4 in kthread ()
  41   [kworker/0:2] (TGID:48) 0xc00412a4 in kthread ()
  40   [kpsmoused] (TGID:47) 0xc00412a4 in kthread ()
  39   [kworker/u4:2] (TGID:40) 0xc00412a4 in kthread ()
  38   [kworker/u4:1] (TGID:39) 0xc00412a4 in kthread ()
  37   [scsi_eh_0] (TGID:38) 0xc00412a4 in kthread ()
  36   [irq/205-hdmi_ir] (TGID:37) 0xc00412a4 in kthread ()
  35   [irq/208-vsync-m] (TGID:36) 0xc00412a4 in kthread ()
  34   [irq/207-vsync-m] (TGID:35) 0xc00412a4 in kthread ()
  33   [B2R2] (TGID:34)  0xc00412a4 in kthread ()
  32   [B2R2_CTL] (TGID:33) 0xc00412a4 in kthread ()
  31   [crypto] (TGID:27) 0xc00412a4 in kthread ()
  30   [nfsiod] (TGID:26) 0xc00412a4 in kthread ()
  29   [fsnotify_mark] (TGID:25) 0xc00412a4 in kthread ()
  28   [kswapd0] (TGID:24) 0xc00412a4 in kthread ()
  27   [rpciod] (TGID:23) 0xc00412a4 in kthread ()
  26   [khubd] (TGID:22) 0xc00412a4 in kthread ()
  25   [ata_sff] (TGID:21) 0xc00412a4 in kthread ()
  24   [kblockd] (TGID:20) 0xc00412a4 in kthread ()
  23   [bioset] (TGID:19) 0xc00412a4 in kthread ()
  22   [writeback] (TGID:18) 0xc00412a4 in kthread ()
  21   [kworker/1:1] (TGID:17) 0xc00412a4 in kthread ()
  20   [kworker/0:1] (TGID:16) 0xc00412a4 in kthread ()
  19   [kdevtmpfs] (TGID:15) 0xc00412a4 in kthread ()
  18   [khelper] (TGID:14) 0xc00412a4 in kthread ()
  17   [kworker/1:0H] (TGID:13) 0xc00412a4 in kthread ()
  16   [kworker/1:0] (TGID:12) 0xc00412a4 in kthread ()
  15   [ksoftirqd/1] (TGID:11) 0xc00412a4 in kthread ()
  14   [migration/1] (TGID:10) 0xc00412a4 in kthread ()
  13   [rcu_sched] (TGID:9) 0xc00412a4 in kthread ()
  12   [rcu_bh] (TGID:8) 0xc00412a4 in kthread ()
  11   [migration/0] (TGID:7) 0xc00412a4 in kthread ()
  10   [kworker/u4:0] (TGID:6) 0xc00412a4 in kthread ()
  9    [kworker/0:0H] (TGID:5) 0xc00412a4 in kthread ()
  8    [kworker/0:0] (TGID:4) 0xc00412a4 in kthread ()
  7    [ksoftirqd/0] (TGID:3) 0xc00412a4 in kthread ()
  6    [kthreadd] (TGID:2) ret_from_fork () at sources/linux-sti/arch/arm/kernel/entry-
common.S:92
  5    systemd (TGID:1)  0xc011619c in sys_epoll_wait () at sources/linux-sti/fs
/eventpoll.c:1605
  4    [swapper/1] (TGID:0 <C1>) cpu_v7_do_idle () at /sources/linux-sti/arch/arm/mm/proc-
v7.S:74
* 3    [swapper/0] (TGID:0 <C0>) cpu_v7_do_idle () at /sources/linux-sti/arch/arm/mm/proc-
v7.S:74
```

8 CPU register commands

(gdb) info registers *- (abbreviation **i r**) Prints the names and values of all registers except floating-point and vector registers (in the selected stack frame)*

(gdb) info all-registers *- (abbreviation **i all-registers**) Prints the names and values of all registers, including floating-point and vector registers (in the selected stack frame)*

(gdb) info registers *<regname> - (abbreviation **i r** <regname>) Prints the value of the register specified by **regname***

For example:

```
(gdb) info registers
r0             0xc1917a58        3247536728
r1             0x0        0
r2             0xbb4e        47950
r3             0x0        0
r4             0xc08544b8        3229959352
r5             0xc084c000        3229925376
r6             0xc0854454        3229959252
r7             0xc089f04a        3230265418
r8             0xc05ed6ac        3227440812
r9             0xc084c000        3229925376
r10            0xc089f04a        3230265418
r11            0xc084c000        3229925376
r12            0x0        0
sp             0xc084dfb0        0xc084dfb0
lr             0xc000f748        0xc000f748 <arch_cpu_idle+40>
pc             0xc001dc68        0xc001dc68 <cpu_v7_do_idle+8>
cpsr           0x600f0093        1611595923
```

9 Variable monitoring commands

- The **print** command (*abbreviation **p***) prints the value of a given **variable** in a **format** appropriate to its data type. You can choose various formats specified by **/f**, where **f*** is a letter defining the format:

  **x: hexadecimal, d: signed decimal, u: unsigned decimal, a: address, c: character, f:float, s: string*

```
(gdb) print [/f] <variable>
```

For example:

```
(gdb) print caps
$21 = (GstCaps *) 0xb4a23bb8
```

- The **display** command adds a variable to the list of variables automatically displayed so that the GDB prints its value each time your program stops

```
(gdb) display <variable>
```

10 Memory analyzing commands

- The command below can be used to analyze the memory in various formats, independently of your program data types:

```
(gdb) x[/nfu] <addr>
```

Where:

- **addr**: first address displayed
- **n, f, and u**: optional parameters that specify the memory size to display and how to format it

  n: repeat count in decimal format (default value is 1). It specifies the memory size to display (counting by units u).

  f: display format. The formats are the same as for the print command (‘x’, ‘d’, ‘u’, ‘o’, ‘t’, ‘a’, ‘c’, ‘f’, ‘s’), plus ‘i’ (for machine instructions). Default format is ‘x’ (hexadecimal). The default format changes each time either **x** or **print** is used.

  u: unit size, which can be

  b::Bytes.

  h::Half-words (two bytes).

  w::Words (four bytes). This is the initial default value.

  g::Giant words (eight bytes).

For example:

- To display a string:

```
(gdb) x/s 0xf017826c
```

- To display a memory area

```
100 bytes in hexadecimal:
(gdb) x/100xb 0xf017826c

100 bytes in decimal
(gdb) x/100db 0xf017826c

100 words in hexadecimal
(gdb) x/100xw 0xf017826c
```

- To avoid disabling the MMU, read a physical memory area by using the *monitor* command (*Note: this is valid only when using the OpenOCD debugger interface, so **not for user space application debug**, see Sending commands to the debugger paragraph*)

```
# For example, read 20 words (32 bits) at address 0x40000400
(gdb) monitor mdw phys 0x40000400 20
0x40000400: 00000080 030ce003 2127d801 4f31e7ee f8df2600 f8df80d4 686390d4 075a685b
0x40000420: b176d503 f0004640 6820f8f9 f928f000 b1684605 48284601 f8f0f000 f0004628
0x40000440: 00000000 00000000 00000000 00000000
```

Bytes (8-bit data) can be read by using *mdb*, and half-words (16-bit data) by using *mdh*.

11 Variable setting command

```
(gdb) set var <variable_name>=<new value>
```

12 Debugging running process commands

The **attach** command can be used to debug an already running process.

*Note: Use either the ps utility or the 'jobs -l' shell command find out the **process-id** of a Unix process.*

```
(gdb) attach <process-id>
```

When the attached process debbugging is complete, you can use the **detach** command to release it from GDB control:

```
(gdb) detach
```

13 Disassemble command

Disassembles a specified section of memory.

(gdb) disassemble *- (abbreviation **disas**) Disassemble the function surrounding the pc of the selected frame.*

(gdb) disassemble *<Function> - (abbreviation **disas** <Function>) Disassemble the given entire function*

(gdb) disassemble *<Address> - (abbreviation **disas** <Address>) Disassemble the address inside a function*

(gdb) disassemble *<Start>,<End> - (abbreviation **disas** <Start>,<End>) Disassemble the specified address range*

(gdb) disassemble *<Function>,+<Length> - (abbreviation **disas** <Function>,+<Length>) Disassemble from the start address of the function with the amount of bytes lenght*

(gdb) disassemble *<Address>,+<Length> - (abbreviation **disas** <Function>,+<Length>) Disassemble from the given start address in a function with the amount of bytes lenght*

(gdb) disassemble /m *[...] - (abbreviation **disas** /m [...]) When this option is specified, the disassemble command shows the source lines that correspond to the disassembled instructions*

(gdb) disassemble /r *[...] - (abbreviation **disas** /r [...]) When this option is specified, the disassemble command shows the raw byte values of all disassembled instructions*

14 Quit (or q) command

```
(gdb) quit - (abbreviation q) Quits gdb
```

15 Sending commands to the debugger

The **monitor** command allows sending arbitrary commands directly to the remote monitor as OpenOCD debugger.

This command does not work when debugging user space applications that use gdbsever interface instead of OpenOCD debugger path.

In that case, since the GDB does not care about the commands it sends, the **monitor** command allows to extend the GDB. You can add new commands that only the external monitor understands and implements.

```
(gdb) monitor <cmd>
```

To get the list of *cmd* commands:

```
(gdb) monitor help
```

16 Reference

- https://sourceware.org/gdb/current/onlinedocs/gdb/

GNU dedugger, a portable debugger that runs on many Unix-like systems

Application binary interface. (In computer software, an application binary interface (ABI) describes the low-level interface between a computer program and the operating system or another program.)

Read Only

Central processing unit

Memory Management Unit. (A hardware device or circuit that supports virtual memory and paging by translating virtual addresses into physical addresses.)