# Examining the Stack

Each time your program performs a function call, the information about where in your program the call was made from is saved in a block of data called a stack frame. The frame also contains the arguments of the call and the local variables of the function that was called. All the stack frames are allocated in a region of memory called the **call stack**.

When your program stops, GDB automatically selects the currently executing frame and describes it briefly as the frame command does (see section Information about a frame).

## Stack frames

The call stack is divided up into contiguous pieces called stack frames, or frames for short; each frame is the data associated with one call to one function. The frame contains the arguments given to the function, the function's local variables, and the address at which the function is executing.

Frame of function main is called the initial frame or the outermost frame. The frame for the function in which execution is actually occurring is called the innermost frame.

Inside your program, stack frames are identified by their addresses. Usually this address is kept in a register called the frame pointer register while execution is going on in that frame.

GDB assigns numbers to all existing stack frames, starting with zero for the innermost frame, one for the frame that called it, and so on upward. These numbers do not really exist in your program; they are assigned by GDB to give you a way of designating stack frames in GDB commands.

Some compilers provide a way to compile functions so that they operate without stack frames. (For example, the gcc option `-fomit-frame-pointer' generates functions without a frame.) This is occasionally done with heavily used library functions to save the frame setup time. GDB has limited facilities for dealing with these function invocations. If the innermost function invocation has no stack frame, GDB nevertheless regards it as though it had a separate frame, which is numbered zero as usual, allowing correct tracing of the function call chain. However, GDB has no provision for frameless functions elsewhere in the stack.

## Backtraces

A backtrace is a summary of how your program got where it is. It shows one line per frame, for many frames, starting with the currently executing frame (frame zero), followed by its caller (frame one), and on up the stack.

### backtrace

### bt

Print 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 C-c.

### backtrace n

### bt n

Similar, but print only the innermost n frames

### backtrace -n

### bt -n

Similar, but print only the outermost n frames

### where

### info stack

### info s

additional aliases for backtrace.

Each line in the backtrace shows the frame number and the function name. The program counter value is also shown--unless you use set print address off. The backtrace also shows the source file name and line number, as well as the arguments to the function. The program counter value is omitted if it is at the beginning of the code for that line number.

Here is an example of a backtrace. It was made with the command `bt 3', so it shows the innermost three frames.

*#0 m4\_traceon (obs=0x24eb0, argc=1, argv=0x2b8c8)*

*at builtin.c:993*

*#1 0x6e38 in expand\_macro (sym=0x2b600) at macro.c:242*

*#2 0x6840 in expand\_token (obs=0x0, t=177664, td=0xf7fffb08)*

*at macro.c:71*

*(More stack frames follow...)*

The display for frame zero does not begin with a program counter value, indicating that your program has stopped at the beginning of the code for line 993 of builtin.c.

## Selecting a frame

Here are the commands for selecting a stack frame; all of them finish by printing a brief description of the stack frame just selected.

### frame n

### f n

Select frame number n. Recall that frame zero is the innermost (currently executing) frame, frame one is the frame that called the innermost one, and so on. The highest-numbered frame is the one for main.

### frame addr

### f addr

Select the frame at address addr. This is useful mainly if the chaining of stack frames has been damaged by a bug, making it impossible for GDB to assign numbers properly to all frames. In addition, this can be useful when your program has multiple stacks and switches between them.

* On the SPARC architecture, frame needs two addresses to select an arbitrary frame: a frame pointer and a stack pointer.
* On the MIPS and Alpha architecture, it needs two addresses: a stack pointer and a program counter.
* On the 29k architecture, it needs three addresses: a register stack pointer, a program counter, and a memory stack pointer.

### up n

Move n frames up the stack. For positive numbers n, this advances toward the outermost frame, to higher frame numbers, to frames that have existed longer. n defaults to one.

### down n

Move n frames down the stack. For positive numbers n, this advances toward the innermost frame, to lower frame numbers, to frames that were created more recently. n defaults to one. You may abbreviate down as do.

All of these commands end by printing two lines of output describing the frame. The first line shows the frame number, the function name, the arguments, and the source file and line number of execution in that frame. The second line shows the text of that source line.

*(gdb) up*

*#1 0x22f0 in main (argc=1, argv=0xf7fffbf4, env=0xf7fffbfc)*

*at env.c:10*

*10 read\_input\_file (argv[i]);*

After such a printout, the list command with no arguments prints ten lines centered on the point of execution in the frame. See section Printing source lines.

### up-silently n

### down-silently n

These two commands are variants of up and down, respectively; they differ in that they do their work silently, without causing display of the new frame. They are intended primarily for use in GDB command scripts, where the output might be unnecessary and distracting.