gdb Commands (1)

Certainly! Here's a list of some common GDB (GNU Debugger) commands and their uses. GDB is a powerful tool for debugging programs written in C, C++, and other languages, allowing developers to see what is going on 'inside' a program while it executes or what it was doing at the moment it crashed.

Basic Commands

- `run` (or `r`): Starts your program in GDB. You can pass arguments to your program after the `run` command.
- `break` (or `b`): Sets a breakpoint at a specified location in your code. This location could be a line number, function name, or address. Execution will stop when it reaches this point.
- `continue` (or `c`): Continues running your program after a breakpoint is hit.
- `next` (or `n`): Runs the next line of code in the current function, stepping over function calls.
- `step` (or `s`): Runs the next line of code, stepping into function calls.
- `list` (or `l`): Lists the source code. By default, it shows ten lines around the current line or the last line displayed.
- `print` (or `p`): Prints the value of an expression in the current context. Variables, register values, or any valid expressions can be printed.
- `watch`: Sets a watchpoint for a variable. A watchpoint stops execution whenever the variable's value changes.
- `info`: Provides information about various aspects of the debugger state, such as `info breakpoints`, `info registers`, etc.
- `backtrace` (or `bt`): Displays the call stack, showing the function call sequence that led to the current point of execution.
- `finish`: Runs until the current function is finished and then stops.
- `quit` (or `q`): Exits GDB.

Advanced Commands

- `frame` (or `f`): Selects and displays a particular frame from the call stack.
- `set var`: Changes the value of a variable.
- `disable`/`enable`: Disables or enables breakpoints and watchpoints.
- `attach`: Attaches GDB to a running process, allowing you to debug it.
- `detach`: Detaches GDB from the currently debugged process, allowing it to continue running.
- `tui enable`: Enables the Text User Interface, providing a more visual way to navigate your code and see the source.
- `record`: Starts recording the execution of your program for reversible debugging.
- `reverse-step` (or `rs`): Steps backward, useful in conjunction with the `record` command.

Conditional Breakpoints

- You can also set conditional breakpoints using `break` if you want the program to stop only if certain conditions are met. For example, `break if i==10` stops at the breakpoint only if the variable `i` equals 10.

Automation

- `source`: Executes GDB commands from a file.
- `define`: Allows you to create your own GDB commands composed of other GDB commands.

GDB is highly customizable and includes many more commands and options. The commands listed here represent some of the most commonly used ones that can help with basic debugging tasks. For a comprehensive list and detailed descriptions, you might want to consult the GDB manual or help system (`help` command in GDB).