# Examining Source Files

GDB can print parts of your program's source, since the debugging information recorded in the program tells GDB what source files were used to build it.

## Printing source lines

To print lines from a source file, use the list command (abbreviated l).

Here are the forms of the list command most commonly used:

### list linenum

Print lines centered around line number linenum in the current source file.

### list function

Print lines centered around the beginning of function function.

### list

Print more lines. If the last lines printed were printed with a list command, this prints lines following the last lines printed; however, if the last line printed was a solitary line printed as part of displaying a stack frame (see section Examining the Stack), this prints lines centered around that line.

### list -

Print lines just before the lines last printed.

By default, GDB prints ten source lines with any of these forms of the list command. You can change this using set listsize:

### set listsize count

Make the list command display count source lines (unless the list argument explicitly specifies some other number).

### show listsize

Display the number of lines that list prints.

Repeating a list command with RET discards the argument, so it is equivalent to typing just list. This is more useful than listing the same lines again. An exception is made for an argument of `-'; that argument is preserved in repetition so that each repetition moves up in the source file.

### list linespec

Print lines centered around the line specified by linespec.

### list first,last

Print lines from first to last. Both arguments are linespecs.

### list ,last

Print lines ending with last.

### list first,

Print lines starting with first.

### list +

Print lines just after the lines last printed.

### list -

Print lines just before the lines last printed.

### list

As described in the preceding table.

Here are the ways of specifying a single source line--all the kinds of linespec.

### number

Specifies line number of the current source file. When a list command has two linespecs, this refers to the same source file as the first linespec.

### +offset

Specifies the line offset lines after the last line printed. When used as the second linespec in a list command that has two, this specifies the line offset lines down from the first linespec.

### -offset

Specifies the line offset lines before the last line printed.

### filename:number

Specifies line number in the source file filename.

### function

Specifies the line of the open-brace that begins the body of the function function.

### filename:function

Specifies the line of the open-brace that begins the body of the function function in the file filename. You only need the file name with a function name to avoid ambiguity when there are identically named functions in different source files.

### \*address

Specifies the line containing the program address address. address may be any expression.

## Searching source files

There are two commands for searching through the current source file for a regular expression.

### forward-search regexp

### search regexp

### fo regexp

The command `forward-search regexp' checks each line, starting with the one following the last line listed, for a match for regexp. It lists the line that is found. You can use synonym `search regexp' or abbreviate the command name as fo.

### reverse-search regexp

The command `reverse-search regexp' checks each line, starting with the one before the last line listed and going backward, for a match for regexp. It lists the line that is found. You can abbreviate this command as rev.

## Specifying source directories

GDB has a list of directories to search for source files; this is called the source path. Each time GDB wants a source file, it tries all the directories in the list, in the order they are present in the list, until it finds a file with the desired name. Note that the executable search path is not used for this purpose. Neither is the current working directory, unless it happens to be in the source path.

If GDB cannot find a source file in the source path, and the object program records a directory, GDB tries that directory too. If the source path is empty, and there is no record of the compilation directory, GDB looks in the current directory as a last resort.

Whenever you reset or rearrange the source path, GDB clears out any information it has cached about where source files are found and where each line is in the file.

When you start GDB, its source path is empty. To add other directories, use the directory command.

### directory dirname ...

Add directory dirname to the front of the source path. Several directory names may be given to this command, separated by `:' or whitespace. You may specify a directory that is already in the source path; this moves it forward, so GDB searches it sooner.

You can use the string `$cdir' to refer to the compilation directory (if one is recorded), and `$cwd' to refer to the current working directory. `$cwd' is not the same as `.'---the former tracks the current working directory as it changes during your GDB session, while the latter is immediately expanded to the current directory at the time you add an entry to the source path.

### directory

Reset the source path to empty again. This requires confirmation.

### show directories

Print the source path: show which directories it contains.

If your source path is cluttered with directories that are no longer of interest, GDB may sometimes cause confusion by finding the wrong versions of source. You can correct the situation as follows:

* Use directory with no argument to reset the source path to empty.
* Use directory with suitable arguments to reinstall the directories you want in the source path. You can add all the directories in one command.

## Source and machine code

You can use the command info line to map source lines to program addresses (and vice versa), and the command disassemble to display a range of addresses as machine instructions.

### info line linespec

Print the starting and ending addresses of the compiled code for source line linespec. You can specify source lines in any of the ways understood by the list command (see section Printing source lines).

For example, we can use info line to discover the location of the object code for the first line of function m4\_changequote:

*(gdb) info line m4\_changecom*

*Line 895 of "builtin.c" starts at pc 0x634c and ends at 0x6350.*

We can also inquire (using \*addr as the form for linespec) what source line covers a particular address:

*(gdb) info line \*0x63ff*

*Line 926 of "builtin.c" starts at pc 0x63e4 and ends at 0x6404.*

After info line, the default address for the x command is changed to the starting address of the line, so that `x/i' is sufficient to begin examining the machine code (see section Examining memory). Also, this address is saved as the value of the convenience variable $\_ (see section Convenience variables).

### disassemble

This specialized command dumps a range of memory as machine instructions. The default memory range is the function surrounding the program counter of the selected frame. A single argument to this command is a program counter value; GDB dumps the function surrounding this value. Two arguments specify a range of addresses (first inclusive, second exclusive) to dump.

We can use disassemble to inspect the object code range shown in the last info line example (the example shows SPARC machine instructions):

*(gdb) disas 0x63e4 0x6404*

*Dump of assembler code from 0x63e4 to 0x6404:*

*0x63e4 <builtin\_init+5340>: ble 0x63f8 <builtin\_init+5360>*

*0x63e8 <builtin\_init+5344>: sethi %hi(0x4c00), %o0*

*0x63ec <builtin\_init+5348>: ld [%i1+4], %o0*

*0x63f0 <builtin\_init+5352>: b 0x63fc <builtin\_init+5364>*

*0x63f4 <builtin\_init+5356>: ld [%o0+4], %o0*

*0x63f8 <builtin\_init+5360>: or %o0, 0x1a4, %o0*

*0x63fc <builtin\_init+5364>: call 0x9288 <path\_search>*

*0x6400 <builtin\_init+5368>: nop*

*End of assembler dump.*