GDB: Complete Debugging Guide

1. Introduction to gdb

What is gdb?

gdb (GNU Debugger) is a powerful debugging tool for C/C++ programs[86]. It allows you to: - Step through code line by line - Watch variable values interactively - Set breakpoints to pause execution - Analyze program state when errors occur - No need to write diagnostic printf statements[86]

Why Use a Debugger?

Types of errors debuggers help with:[86] - Compilation errors (caught by compiler) - Logical errors (program runs but wrong output) - Runtime errors (segmentation faults, crashes)

Remember: The compiler is never faulty. Question your understanding and code first[86].

---

2. Starting gdb

Compilation with Debug Symbols

Compile with -g flag:[86]

```
$ gcc -Wall -g tarea.c
```

The `-g` flag includes debugging symbols in the executable, allowing gdb to show source code and variable names.

Example program (tarea.c):[86]

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int x1, y1, x2, y2, x3, y3;
    double area;

    printf("Program to calculate the area of a triangle\n");
    printf("Enter the coordinates of the first corner: ");
    scanf("%d%d", &x1, &y1);
    printf("Enter the coordinates of the second corner: ");
    scanf("%d%d", &x2, &y2);
    printf("Enter the coordinates of the third corner: ");
    scanf("%d%d", &x3, &y3);
    area = abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0;

    printf("The area of the triangle = %lf\n", area);
    exit(0);
}
```

Starting gdb

Launch gdb:[86]

```
$ gdb ./a.out
```

gdb prompt:

```
GNU gdb (GDB) 8.1.1
...
Reading symbols from ./a.out...done.
(gdb)
```

---

3. Basic Navigation Commands

list Command - View Source Code

Basic list:[86]

```
(gdb) list
```

Shows 10 lines at a time, starting from main function.

Output example:[86]

```
1    #include <stdio.h>
2    #include <stdlib.h>
3
4    int main()
5    {
6        int x1, y1, x2, y2, x3, y3;
7        double area;
8
9        printf("Program to calculate the area of a triangle\n");
10       printf("Enter the coordinates of the first corner: ");
```

Subsequent list commands:[86]

```
(gdb) list
```

Shows next 10 lines.

List specific lines:[86]

```
(gdb) list 18,22
```

Output:

```
18        printf("Enter the coordinates of the third corner: ");
19        scanf("%d%d", &x3, &y3);
20
21        area = abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0;
22
```

List around a line:[86]

```
(gdb) list 18
```

Shows 10 lines centered around line 18.

List specific function:[86]

```
(gdb) list main
(gdb) list function_name
```

Repeating Commands

Press Enter/Return:[86] Repeats the last command. Very useful with `list`, `step`, `next`.

---

4. Running Programs in gdb

run Command - Start Execution

Basic run:[86]

```
(gdb) run
```

Output:[86]

```
Starting program: /home/user/a.out
Program to calculate the area of a triangle
Enter the coordinates of the first corner: 0 0
Enter the coordinates of the second corner: 4 0
Enter the coordinates of the third corner: 2 3
The area of the triangle = 6.000000
[Inferior 1 (process 12345) exited normally]
```

Run with command-line arguments:[86]

```
(gdb) run arg1 arg2 arg3
```

Run with input redirection:[86]

```
(gdb) run < inputfile.txt
```

---

5. Breakpoints

Setting Breakpoints

Break at line number:[86]

```
(gdb) break 18
```

Output:

```
Breakpoint 1 at 0x400632: file tarea.c, line 18.
```

Break at function:[86]

```
(gdb) break main
```

Output:

```
Breakpoint 2 at 0x400580: file tarea.c, line 6.
```

Break at function in specific file:[86]

```
(gdb) break filename.c:function_name
(gdb) break filename.c:25
```

Viewing Breakpoints

List all breakpoints:[86]

```
(gdb) info break
```

Output:

```
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400632 in main at tarea.c:18
2       breakpoint     keep y   0x0000000000400580 in main at tarea.c:6
```

Managing Breakpoints

Disable breakpoint:[86]

```
(gdb) disable 1
```

Enable breakpoint:[86]

```
(gdb) enable 1
```

Delete breakpoint:[86]

```
(gdb) delete 1
```

Delete all breakpoints:[86]

```
(gdb) delete
```

Running to Breakpoint

Continue execution:[86]

```
(gdb) run
Starting program: /home/user/a.out
Program to calculate the area of a triangle

Breakpoint 1, main () at tarea.c:6
6        int x1, y1, x2, y2, x3, y3;
```

Continue to next breakpoint:[86]

```
(gdb) continue
```

Output:

```
Continuing.
Enter the coordinates of the first corner: 0 0
Enter the coordinates of the second corner: 4 0

Breakpoint 2, main () at tarea.c:18
18       printf("Enter the coordinates of the third corner: ");
```

---

6. Stepping Through Code

next Command - Execute Next Line

Execute current line, stay in same function:[86]

```
(gdb) next
```

Short form:

```
(gdb) n
```

Example execution:[86]

```
18       printf("Enter the coordinates of the third corner: ");
(gdb) next
Enter the coordinates of the third corner: 19       scanf("%d%d", &x3, &y3);
```

step Command - Step Into Functions

Step into function calls:[86]

```
(gdb) step
```

Short form:

```
(gdb) s
```

Difference from next: - `next`: Executes function call and stops at next line - `step`: Enters the function and stops at first line inside

finish Command - Run Until Return

Complete current function:[86]

```
(gdb) finish
```

Runs until current function returns, then stops.

Example:

```
(gdb) finish
Run till exit from #0  factorial (n=5) at factorial.c:10
0x0000000000400632 in main () at factorial.c:25
25          result = factorial(num);
Value returned is $1 = 120
```

return Command - Force Return

Return immediately from function:[86]

```
(gdb) return
(gdb) return value
```

Forces immediate return, skipping remaining code in function.

---

7. Examining Variables

print Command - Display Values

Print variable:[86]

```
(gdb) print x1
```

Output:

```
$1 = 0
```

Print expression:[86]

```
(gdb) print x1 + x2
```

Output:

```
$2 = 4
```

Print with different formats:[86]

```
(gdb) print x1          # Decimal (default)
(gdb) print/x x1        # Hexadecimal
(gdb) print/o x1        # Octal
```

```
(gdb) print/t x1        # Binary
(gdb) print/c x1        # Character
```

Value history:[86] Results are stored as $1, $2, $3, etc.

```
(gdb) print $1 + $2
```

display Command - Auto-Display Variables

Automatically display after each step:[86]

```
(gdb) display area
```

Output:

```
1: area = 0
```

After each `next` or `step`, gdb shows:

```
1: area = 6
```

View all displays:[86]

```
(gdb) info display
```

Remove display:[86]

```
(gdb) undisplay 1
```

Disable/enable display:[86]

```
(gdb) disable display 1
(gdb) enable display 1
```

set Command - Modify Variables

Change variable value:[86]

```
(gdb) set var x1 = 10
(gdb) set var area = 0.0
```

Example:[86]

```
(gdb) print area
$3 = 5.5
(gdb) set var area = 10.0
(gdb) print area
$4 = 10
```

8. Watchpoints

watch Command - Break on Variable Change

Set watchpoint:[86]

```
(gdb) watch x1
```

Output:

```
Hardware watchpoint 3: x1
```

Program stops whenever `x1` changes value.

View watchpoints:[86]

```
(gdb) info watch
```

Delete watchpoint:[86]

```
(gdb) delete 3
```

Example output when watchpoint triggers:[86]

```
Hardware watchpoint 3: x1
Old value = 0
New value = 5
main () at tarea.c:12
12       printf("Enter the coordinates of the second corner: ");
```

---

9. Conditional Breakpoints

Setting Conditions on Breakpoints

Break only if condition true:[86]

```
(gdb) break 234 if p == 0
```

Stops at line 234 only when pointer `p` is NULL.

Add condition to existing breakpoint:[86]

```
(gdb) condition 2 i == 100
```

Breakpoint 2 now triggers only when `i` equals 100.

Remove condition:[86]

```
(gdb) condition 2
```

Example - Finding NULL pointer:[86]

```
(gdb) break 234 if p == 0
Breakpoint 1 at 0x400632: file prog.c, line 234.
(gdb) run
```

```
...
Breakpoint 1, main () at prog.c:234
234       p->data = value;
(gdb) print p
$1 = (node *) 0x0
```

ignore Command - Skip Breakpoint Hits

Ignore next N hits:[86]

```
(gdb) ignore 2 5
```

Breakpoint 2 will be ignored for the next 5 hits.

Example:[86]

```
(gdb) break 15
Breakpoint 1 at 0x400580: file loop.c, line 15.
(gdb) ignore 1 99
Will ignore next 99 crossings of breakpoint 1.
(gdb) run
...
Breakpoint 1, main () at loop.c:15  # Stops on 100th iteration
15        sum += i;
```

---

10. Call Stack and Frames

backtrace Command - View Call Stack

Show call stack:[86]

```
(gdb) backtrace
(gdb) bt
```

Output example:[86]

```
#0  factorial (n=3) at factorial.c:10
#1  0x0000000000400625 in factorial (n=4) at factorial.c:12
#2  0x0000000000400625 in factorial (n=5) at factorial.c:12
#3  0x0000000000400655 in main () at factorial.c:25
```

Shows function call hierarchy from current function to main.

frame Command - Navigate Frames

Show current frame:[86]

```
(gdb) frame
```

Output:

```
#0  factorial (n=3) at factorial.c:10
10       if (n <= 1) return 1;
```

Switch to specific frame:[86]

```
(gdb) frame 2
```

Output:

```
#2  0x0000000000400625 in factorial (n=5) at factorial.c:12
12       return n * factorial(n - 1);
```

Move up/down stack:[86]

```
(gdb) up        # Move to calling function
(gdb) down      # Move to called function
```

Get frame info:[86]

```
(gdb) info frame
```

Shows detailed information about current frame.

---

11. Memory Examination

x Command - Examine Memory

Examine memory at address:[86]

```
(gdb) x/5wx A
```

Format: x/[count][format][size] address

Sizes:[86] - `b` - byte (1 byte) - `h` - halfword (2 bytes) - `w` - word (4 bytes) - `g` - giant (8 bytes)

Formats:[86] - `x` - hexadecimal - `d` - decimal - `u` - unsigned decimal - `o` - octal - `t` - binary - `c` - character - `s` - string

Examples:[86]

```
(gdb) x/5wx A           # 5 words in hex starting at A
(gdb) x/1wx &i          # 1 word in hex at address of i
(gdb) x/10bd array      # 10 bytes in decimal from array
(gdb) x/s str           # String at str
```

Output example:[86]

```
(gdb) x/5wx A
0x7fffffffe420: 0x00000001  0x00000002  0x00000003  0x00000004
0x7fffffffe430: 0x00000005
```

---

12. Advanced Features

Working with Arrays

Print array:[86]

```
(gdb) print A[0]@10
```

Prints 10 elements starting from A[0].

Example:[86]

```
(gdb) print A[0]@5
$1 = {1, 2, 3, 4, 5}
```

Working with Pointers

Print dereferenced pointer:[86]

```
(gdb) print *ptr
```

Print pointer address:[86]

```
(gdb) print ptr
```

Print structure through pointer:[86]

```
(gdb) print *node_ptr
```

Type Information

Check variable type:[86]

```
(gdb) ptype variable
```

Example:[86]

```
(gdb) ptype area
type = double
(gdb) ptype x1
type = int
```

---

13. Multi-File Debugging

Debugging Programs with Multiple Files

Scenario 1: Files included with #include:[86]

Compile:

```
$ gcc -Wall -g allparts.c
```

In gdb, list specific file:[86]

```
(gdb) list part1.c:1
```

Scenario 2: Separately compiled files:[86]

Compile:

```
$ gcc -Wall -g -c part1.c
$ gcc -Wall -g -c part2.c
$ gcc -Wall -g -c allparts.c
$ gcc -g -o a.out allparts.o part1.o part2.o
```

In gdb:[86]

```
$ gdb ./a.out
(gdb) list part1.c:1
(gdb) list part2.c:function_name
(gdb) break part1.c:25
(gdb) break part2.c:function_name
```

Loading Different Executable

Load new executable without restarting gdb:[86]

```
(gdb) file newprog
```

Useful when you recompile and want to debug new version.

---

14. Help System

Getting Help in gdb

General help:[86]

```
(gdb) help
```

Help on specific command:[86]

```
(gdb) help break
(gdb) help print
(gdb) help run
```

Search help topics:[86]

```
(gdb) apropos keyword
```

Example:[86]

```
(gdb) apropos breakpoint
```

Lists all commands related to breakpoints.

List command categories:[86]

```
(gdb) help all
```

---

15. Quitting gdb

Exiting gdb Session

Quit gdb:[86]

```
(gdb) quit
(gdb) q
```

If program is running:

```
A debugging session is active.
Inferior 1 [process 12345] will be killed.
Quit anyway? (y or n) y
```

---

16. Common Debugging Scenarios

Example 1: Segmentation Fault

Program crashes:

```
$ ./a.out
Segmentation fault (core dumped)
```

Debug:[86]

```
$ gdb ./a.out
(gdb) run
...
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400632 in main () at prog.c:234
234         p->data = value;
(gdb) print p
$1 = (node *) 0x0
```

p is NULL, causing segfault.

Example 2: Infinite Loop

Set breakpoint in loop:[86]

```
(gdb) break 45
(gdb) run
...
Breakpoint 1, main () at prog.c:45
45          while (i < n) {
```

```
(gdb) display i
(gdb) display n
(gdb) continue
```

Watch i and n values to identify why loop doesn't terminate.

Example 3: Wrong Calculation

Use print to check intermediate values:[86]

```
(gdb) break 21
(gdb) run
...
Breakpoint 1, main () at tarea.c:21
21         area = abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0;
(gdb) print x1
$1 = 0
(gdb) print y2 - y3
$2 = -3
(gdb) print x1 * (y2 - y3)
$3 = 0
```

Step through calculation to find error.

---

17. Command Summary

Command	Shortcut	Description
`run [args]`	`r`	Start program execution
`break [location]`	`b`	Set breakpoint
`continue`	`c`	Continue to next breakpoint
`next`	`n`	Execute next line (step over)
`step`	`s`	Step into function
`finish`	`-`	Run until function returns
`print [expr]`	`p`	Print value/expression
`display [expr]`	`-`	Auto-display after each step
`watch [var]`	`-`	Break when variable changes
`backtrace`	`bt`	Show call stack
`frame [n]`	`f`	Select stack frame
`list [loc]`	`l`	Show source code
`info break`	`i b`	List breakpoints
`delete [n]`	`d`	Delete breakpoint
`set var x=val`	`-`	Set variable value
`quit`	`q`	Exit gdb
`help [cmd]`	`h`	Get help

This comprehensive guide covers all gdb features from the PDF with practical examples for each command and use case.