

GDB Tutorial

Young W. Lim

2017-02-13 Mon

1 Introduction

"Self-service Linux: Mastering the Art of Problem Determination", Mark Wilding

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

Summary (1)

- Starting and Stopping
 - quit : exit GDB
 - run : run your program
 - kill : stop your program
- Breakpoints
 - break func : set breakpoint at entry to function func
 - break *0x80483c3 : set breakpoint at address 0x80483c3
 - delete 1 : delete breakpoint 1
 - delete : delete all breakpoints

Summary (2)

- Execution

- stepi : execute one instruction
- setpi 4 : execute 4 instructions
- nexti : like stepi, but proceed through function calls
- continue : resume execution
- finish : run until current function returns

- Examining code

- disas : disassemble current function
- disas func : disassemble function func
- disas 0x80483b7 : disassemble function around address 0x80483b7
- disas 0x80483b7 0x80483c7 : disassemble code within specified address range
- print /x \$eip : print program counter in hex

Controlling Processes

- executing a program in the gdb
- attaching a running program to the gdb
- using core dump (process image) in the gdb

Executing a program in the gdb (1)

```
g++ hang.C -g -o hang
gdb hang
(gdb) break main
(gdb) run user // hang user
(gdb) list

(gdb) show args
(gdb) set args 6 7
(gdb) show args
(gdb) run // hang 6 7
```

```
int main(int argc, char *argv[])
{
    getpid();

    if (argc < 2) {
        printf("hang (user|system)");
    }

    ...
}
```

Executing a program in the gdb (2)

```
(gdb) show environment
(gdb) show environment var
(gdb) set environment var=val
(gdb) unset environment var

(gdb) set environment FOO=BAR
(gdb) show environment FOO
(gdb) unset environment FOO
(gdb) show environment FOO
```


Attaching a running program to the gdb (1)

```
hang system // must stop : sleep(5000);  
gdb -3051 // attaching to process 3051  
(gdb) bt // backtrace  
(gdb) up // upward stack  
(gdb) list
```

Attaching a running program to the gdb (2)

```
bt:  backtrace
     #0 nanosleep() from /lib/libc.so.6
     #1 sleep () from /lib/libc.so.6
     #2 main (...) at hang.C ...
     #3 __lib_start_main () from /lib/libc.so.6
up:  upward stack
     #1 sleep () from /lib/libc.so.6
up:  upward stack
     #2 main (...) at hang.C ...
list: displaying the code
     shows main code around sleep( 5000 );
```

core dump file contains

- process heap
- memory image
- memory segments

Making Core Dump File (1)

make_core.C

```
---  
int main(void) {  
    int *p=0;  
    *p = 1;  
    return 0;  
}
```

p is int pointer variable
initialize p with 0, then assign 1 to *p
dereferencing the NULL pointer
incurs trap with SIGSEGV
Segmentation fault

Making Core Dump File (2)

```
make_core.C
```

```
g++ -o make_core make_core.C
```

```
make_core
```

```
Segmentation fault -> core
```

```
gdb make_core core
```

Making Core Dump File (3)

```
ulimit -a  
ulimit -c unlimited  
ls -l core  
  
readelf -all core
```

[ulimit -a] shows
shell resource settings
core file size (blocks, -c) 0
[ulimit -c unlimited]
removes limitation

Core File Name and Location

```
/proc/sys/kernel/core_pattern
```

```
/core_file/%u.%e.%p.core
```

```
%u: process name
```

```
%e: executing user id
```

```
%p: pid
```

```
.500.user.12589.core in core_file directory
```

```
directory /core_files
```

```
echo "/core_file/%u.%e.%p.core"
```

```
> /proc/sys/kernel/core_pattern
```

```
ls -l /core_files
```

Generating a Core File in GDB

```
help generate-core-file
```

```
generate-core-file [core-file-name]
```

```
default core file name: core.<pid>
```


Checking Memory Map

- memory map : memory segment list
- process heap
- process stack
- execution code
- shared libraries
- global var address
- function address

```
info program
```

```
shell cat /proc/<pid>/maps
```

frame 1

print &var

backtrace

bt

bt_full

- backtrace (bt) shows a function list in a stack
- PC addr -> function name
- stack tracing
- #0, #1, : stack frame no
- x86 based
- 0x40000000 shared libraries
- 0x08048000 exec file image
- address mapping :
/proc/<pid>/maps
- bt_full : shows local var's

finish

frame 3

up

down

- finish the curr func in the curr stack frame, return to the caller
- always start with #0 stack frame: the topmost frame
- up : #0 -> #1
- down : #1 -> #0

Stack Frame Information

```
info frame 2
```

```
Stack frame at at 0xbffff330:
```

```
eip = 0x80483e9 in function3 (st_prog.c:16): saved eip 0x804843a  
eip (extended instruction pointer) command to be executed in a frame  
called by frame at 0xbffff370, caller of frame at 0xbffff310  
source language c.
```

```
Arglist at 0xbffff328, args: string=0xbffff340 "This is a local string"
```

```
Locals at 0xbffff328, Previous frame's sp is 0xbffff330
```

```
Saved registers:
```

```
ebp at 0xbffff328, eip at 0xbffff32c
```

- eip (extended instruction pointer)
- command to be executed in a frame
- called by frame
- called of frame

Checking Variables

- global variable
- static variable
- auto variable

```
const char *cstr = "Hello";
```

```
without -g  
print cstr  
$2 = 12345...
```

```
with -g  
printf "%s\n", cstr  
Hello
```

```
info variable sint  
All variables matching  
regular expression "sint":
```

```
print /x sint  
$3 = 0x5
```

Convenience Variable

```
print var  
print /x var  
print $1 + 5
```

```
$num = value
```

```
printf "val in hex is %x\n", var
```

- convenience variable
- gdb automatically creates var \$1, \$2, ...

Array

```
int list[5] = {0,1,2,3,4};  
print list  
print list[3]
```

```
int *array = (int *) malloc(len * sizeof(int));  
array[0]=5;  
array[1]=1;  
array[2]=5;  
print array@3
```

```
whatis a  
type = int
```

```
examine (x)  
x /8xw 0x08048000
```

Formatted Output

cb 1byte ascii char

dh 2byte decimal

dw 4byte decimal

xb 1byte hexadecimal

xh 2byte hexadecimal

xw 4byte hexadecimal

xg 8byte hexadecimal

s string

print cstr

x /s 0x090485c0

printf "%s\n", 0x080485c0

print (char *) 0x080485c0

Type Casting

```
class myC {  
public:  
int var;  
myC() { var=5 }  
};
```

```
print (myC) *0x080485c0
```

```
set variable a=5
```

```
set a=5
```

```
print a
```

```
set $eax=1
```

```
print $eax
```

Register Dump

- eax function return val
- eip command PC
- ebp, esp stack, stack segment
- 32-bit, 64-bit machine difference

```
info registers
```

```
info all-registers
```

```
print $eax
```

```
next <N>  without going inside a function
step <N>  with going inside a function
nexti <N> next for the assembly command
stepi <N> step for the assembly command
continue
jump <address>
until
call <function>
CTRL-C
```

<N> the number of commands (machine or assembly)

```
break main  
run  
next  
step
```

```
show step-mode  
set step-mode on  
show step-mode  
next  
step -> now go into printf()
```

- without debug symbol, step cannot go into printf()
- after setting step-mode on, step goes into printf()

before fork called

```
follow-fork-mode child
```

```
set follow-fork-mode ask
```

```
show follow-fork-mode
```

```
proc 1 fork proc2 & proc3
```

```
proc 2 fork proc4
```

```
info signals SIGALRM
```

- Stop :gdb stops execution, user takes control
- Print : gdb displays messages
- Pass to Programm
- Description

Handling Signals

```
#include <stdio.h>
#include <unistd.h>
#include <sys/utsname.h>
```

```
int main() {
alarm(7);
sleep(9);
return 0;
}
```

```
handle SIGALRM nopass
handle SIGALRM stop
```

```
info signals
```

- run : terminates because no handler exist for SIGALRM
- after "handle SIGALRM nopass", the program exits normally