GDB十分钟教程

GDB十分钟教程

作者：liigo

原文链接：http://blog.csdn.net/liigo/archive/2006/01/17/582231.aspx

日期：2006年1月16日

本文写给主要工作在Windows操作系统下而又需要开发一些跨平台软件的程序员朋友，以及程序爱好者。

GDB是一个由GNU开源组织发布的、UNIX/Linux操作系统下的、基于命令行的、功能强大的程序调试工具。

GDB中的命令固然很多，但我们只需掌握其中十个左右的命令，就大致可以完成日常的基本的程序调试工作。

命令	解释	示例
file <文件名>	加载被调试的可执行程序文件。 因为一般都在被调试程序所在目录下执行GDB，因而文本名不需要带路径。	(gdb) file gdb-sample
r	Run的简写，运行被调试的程序。 如果此前没有下过断点，则执行完整个程序；如果有断点，则程序暂停在第一个可用断点处。	(gdb) r
c	Continue的简写，继续执行被调试程序，直至下一个断点或程序结束。	(gdb) c
b <行号> b <函数名称> b *<函数名称> b *<代码地址> d [编号]	b: Breakpoint的简写，设置断点。两可以使用“行号”“函数名称”“执行地址”等方式指定断点位置。 其中在函数名称前面加“*”符号表示将断点设置在“由编译器生成的prolog代码处”。如果不了解汇编，可以不予理会此用法。 d: Delete breakpoint的简写，删除指定编号的某个	(gdb) b 8 (gdb) b main (gdb) b *main (gdb) b *0x804835c (gdb) d

	断点，或删除所有断点。断点编号从1开始递增。	
s, n	s: 执行一行源程序代码，如果此行代码中有函数调用，则进入该函数； n: 执行一行源程序代码，此行代码中的函数调用也一并执行。 s 相当于其它调试器中的“Step Into (单步跟踪进入)”； n 相当于其它调试器中的“Step Over (单步跟踪)”。 这两个命令必须在有源代码调试信息的情况下才可以使用(GCC编译时使用“-g”参数)。	(gdb) s (gdb) n
si, ni	si命令类似于s命令，ni命令类似于n命令。所不同的是，这两个命令(si/ni)所针对的是汇编指令，而s/n针对的是源代码。	(gdb) si (gdb) ni
p <变量名称>	Print的简写，显示指定变量(临时变量或全局变量)的值。	(gdb) p i (gdb) p nGlobalVar
display ... undisplay <编号>	display，设置程序中断后欲显示的数据及其格式。 例如，如果希望每次程序中断后可以看到即将被执行的下一条汇编指令，可以使用命令 “display /i $pc” 其中 $pc 代表当前汇编指令，/i 表示以十六进行显示。 当需要关心汇编代码时，此命令相当有用。 undispaly，取消先前的display设置，编号从1开始递增。	(gdb) display /i $pc (gdb) undisplay 1
i	Info的简写，用于显示各类信息，详情请查阅“help i”。	(gdb) i r
q	Quit的简写，退出GDB调试环境。	(gdb) q
help [命令名称]	GDB帮助命令，提供对GDB名种命令的解释说明。 如果指定了“命令名称”参数，则显示该命令的详细说明； 如果没有指定参数，则分类显示所有GDB命令，供用户进一步浏览和查询。	(gdb) help display

废话不多说，下面开始实践。

先给出一个示例用的小程序，c语言代码，简单的不能再简单了：

```
1    //此程序仅作为“GDB十分钟教程”的示例代码, by liigo
2    //Email: liigo@sina.com
3    //blog: http://blog.csdn.net/liigo
```

```
4   //WebSite: www.liigo.com
5
6   #include <stdio.h>
7
8   int nGlobalVar = 0;
9
10  int tempFunction(int a, int b)
11  {
12      printf("tempFunction is called, a = %d, b = %d /n", a, b);
13      return (a + b);
14  }
15
16  int main()
17  {
18      int n;
19      n = 1;
20      n++;
21      n--;
22
23      nGlobalVar += 100;
24      nGlobalVar -= 12;
25
26      printf("n = %d, nGlobalVar = %d /n", n, nGlobalVar);
27
28      n = tempFunction(1, 2);
29      printf("n = %d", n);
30
31      return 0;
32  }
33
```

请将此代码复制出来并保存到文件 gdb-sample.c 中，然后切换到此文件所在目录，用GCC编译之：

```
gcc gdb-sample.c -o gdb-sample -g
```

在上面的命令行中，使用 -o 参数指定了编译生成的可执行文件名为 gdb-sample，使用参数 -g 表示将源代码信息编译到可执行文件中。如果不使用参数 -g，会给后面的GDB调试造成不便。当然，如果我们没有程序的源代码，自然也无从使用 -g 参数，调试/跟踪时也只能是汇编代码级别的调试/跟踪。

下面“gdb”命令启动GDB，将首先显示GDB说明，不管它：

```
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu".
(gdb)
```

上面最后一行"(gdb) "为GDB内部命令引导符,等待用户输入GDB命令。

下面使用"file"命令载入被调试程序 gdb-sample(这里的 gdb-sample 即前面 GCC 编译生成的可执行文件):

```
(gdb) file gdb-sample
Reading symbols from gdb-sample...done.
```

上面最后一行提示已经加载成功。

下面使用"r"命令执行(Run)被调试文件,因为尚未设置任何断点,将直接执行到程序结束:

```
(gdb) r
Starting program: /home/liigo/temp/test_jmp/test_jmp/gdb-sample
n = 1, nGlobalVar = 88
tempFunction is called, a = 1, b = 2
n = 3
Program exited normally.
```

下面使用"b"命令在 main 函数开头设置一个断点(Breakpoint):

```
(gdb) b main
Breakpoint 1 at 0x804835c: file gdb-sample.c, line 19.
```

上面最后一行提示已经成功设置断点,并给出了该断点信息:在源文件 gdb-sample.c 第19行处设置断点;这是本程序的第一个断点(序号为1);断点处的代码地址为 0x804835c(此值可能仅在本次调试过程中有效)。回过头去看源代码,第19行中的代码为"n = 1",恰好是 main 函数中的第一个可执行语句(前面的"int n;"为变量定义语句,并非可执行语句)。

再次使用"r"命令执行(Run)被调试程序:

```
(gdb) r
Starting program: /home/liigo/temp/gdb-sample

Breakpoint 1, main () at gdb-sample.c:19
19 n = 1;
```

程序中断在gdb-sample.c第19行处,即main函数是第一个可执行语句处。

上面最后一行信息为:下一条将要执行的源代码为"n = 1;",它是源代码文件gdb-sample.c中的第19行。

下面使用“s”命令(Step)执行下一行代码(即第19行“n = 1;”):

```
(gdb) s
20 n++;
```

上面的信息表示已经执行完“n = 1;”,并显示下一条要执行的代码为第20行的“n++;”。

既然已经执行了“n = 1;”,即给变量 n 赋值为 1,那我们用“p”命令(Print)看一下变量 n 的值是不是 1 :

```
(gdb) p n
$1 = 1
```

果然是 1。($1大致是表示这是第一次使用“p”命令——再次执行“p n”将显示“$2 = 1”——此值

处。)

下面我们分别在第26行、tempFunction 函数开头各设置一个断点(分别使用命令“b 26”“b tempFunction”):

```
(gdb) b 26
Breakpoint 2 at 0x804837b: file gdb-sample.c, line 26.
(gdb) b tempFunction
Breakpoint 3 at 0x804832e: file gdb-sample.c, line 12.
```

使用“c”命令继续(Continue)执行被调试程序,程序将中断在第二个断点(26行),此时全局变量 nGlobalVar 的值应该是 88;再一次执行“c”命令,程序将中断于第三个断点(12行,tempFunction 函数开头处),此时 tempFunction 函数的两个参数 a、b 的值应分别是 1 和 2:

```
(gdb) c
Continuing.

Breakpoint 2, main () at gdb-sample.c:26
26 printf("n = %d, nGlobalVar = %d /n", n, nGlobalVar);
(gdb) p nGlobalVar
$2 = 88
(gdb) c
Continuing.
n = 1, nGlobalVar = 88

Breakpoint 3, tempFunction (a=1, b=2) at gdb-sample.c:12
12 printf("tempFunction is called, a = %d, b = %d /n", a, b);
(gdb) p a
$3 = 1
(gdb) p b
$4 = 2
```

上面反馈的信息一切都在我们预料之中,哈哈~~~

再一次执行“c”命令(Continue),因为后面再也没有其它断点,程序将一直执行到结束:

```
(gdb) c
Continuing.
tempFunction is called, a = 1, b = 2
n = 3
Program exited normally.
```

有时候需要看到编译器生成的汇编代码,以进行汇编级的调试或跟踪,又该如何操作呢?

这就要用到display命令“display /i $pc”了(此命令前面已有详细解释):

```
(gdb) display /i $pc
(gdb)
```

此后程序再中断时,就可以显示出汇编代码了:

```
(gdb) r
Starting program: /home/liigo/temp/test_jmp/test_jmp/gdb-sample

Breakpoint 1, main () at gdb-sample.c:19
19 n = 1;
1: x/i $pc 0x804835c <main+16>: movl $0x1,0xfffffffc(%ebp)
```

看到了汇编代码,“n = 1;”对应的汇编代码是“movl $0x1,0xfffffffc(%ebp)”。

并且以后程序每次中断都将显示下一条汇编指定(“si”命令用于执行一条汇编代码—区别于“s”执行一行C代码):

```
(gdb) si
20 n++;
1: x/i $pc 0x8048363 <main+23>: lea 0xfffffffc(%ebp),%eax
(gdb) si
0x08048366 20 n++;
1: x/i $pc 0x8048366 <main+26>: incl (%eax)
(gdb) si
21 n--;
1: x/i $pc 0x8048368 <main+28>: lea 0xfffffffc(%ebp),%eax
(gdb) si
0x0804836b 21 n--;
1: x/i $pc 0x804836b <main+31>: decl (%eax)
(gdb) si
23 nGlobalVar += 100;
1: x/i $pc 0x804836d <main+33>: addl $0x64,0x80494fc
```

接下来我们试一下命令“b *<函数名称>”。

为了更简明,有必要先删除目前所有断点(使用“d”命令——Delete breakpoint):

```
(gdb) d
Delete all breakpoints? (y or n) y
(gdb)
```

当被询问是否删除所有断点时,输入“y”并按回车键即可。

下面使用命令“b *main”在 main 函数的 prolog 代码处设置断点(prolog、epilog,分别表示编译器在每个函数的开头和结尾自行插入的代码):

```
(gdb) b *main
Breakpoint 4 at 0x804834c: file gdb-sample.c, line 17.
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/liigo/temp/test_jmp/test_jmp/gdb-sample

Breakpoint 4, main () at gdb-sample.c:17
17 {
1: x/i $pc 0x804834c <main>: push %ebp
(gdb) si
0x0804834d 17 {
1: x/i $pc 0x804834d <main+1>: mov %esp,%ebp
(gdb) si
0x0804834f in main () at gdb-sample.c:17
17 {
1: x/i $pc 0x804834f <main+3>: sub $0x8,%esp
(gdb) si
0x08048352 17 {
1: x/i $pc 0x8048352 <main+6>: and $0xfffffff0,%esp
(gdb) si
0x08048355 17 {
1: x/i $pc 0x8048355 <main+9>: mov $0x0,%eax
(gdb) si
0x0804835a 17 {
1: x/i $pc 0x804835a <main+14>: sub %eax,%esp
(gdb) si
19 n = 1;
```

```
1: x/i $pc 0x804835c <main+16>: movl $0x1,0xfffffffc(%ebp)
```

此时可以使用“i r”命令显示寄存器中的当前值——“i r”即“Infomation Register”:

```
(gdb) i r
eax 0xbffff6a4 -1073744220
ecx 0x42015554 1107383636
edx 0x40016bc8 1073834952
ebx 0x42130a14 1108544020
esp 0xbffff6a0 0xbffff6a0
ebp 0xbffff6a8 0xbffff6a8
esi 0x40015360 1073828704
edi 0x80483f0 134513648
eip 0x8048366 0x8048366
eflags 0x386 902
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x33 51
```

当然也可以显示任意一个指定的寄存器值:

```
(gdb) i r eax
eax 0xbffff6a4 -1073744220
```

最后一个要介绍的命令是“q”,退出(Quit)GDB调试环境:

```
(gdb) q
The program is running. Exit anyway? (y or n) y
```


liigo@sina.com

www.liigo.com

http://blog.csdn.net/liigo/

QQ: 175199125

顶 踩
47 1

- 上一篇 我知道AWT/Swing不受欢迎的根本原因了：外观不够漂亮！附图
- 下一篇 《包青天》中的《鸳鸯蝴蝶梦》单元，剧中有一个很漂亮的女子叫“离垢”

相关文章推荐

- 用GDB调试程序（一）
- gdb详解
- codeblocks基本调试方法—gdb—Debugger
- GDB 的进入和退出
- gdb调式程序
- GDB问题汇总
- GDB -x 选项
- GDB 使用——Linux C编程
- gdb参数详解（整理过）
- gdb基本使用方法及常用命令

查看评论

nhdd_csdn 55楼 2017-07-07 13:49发表
十分感谢

爆米花好美啊 54楼 2017-05-27 00:24发表
很强 谢谢

sinat_33006005 53楼 2017-05-08 18:52发表
谢谢分享

qq_22267405 52楼 2017-03-28 13:37发表
什么第一次打开还能看到图片和文章全部内容，之后再打开就好多是空白的。。。黑框里也是空白的。。。T-T

hacker_2017 51楼 2017-03-17 14:45发表
https://gitlore.com/subject/15《100个gdb小技巧》

50楼 2017-03-04 10:40发表

好文章, 转载了, 谢谢~

nusit_305 49楼 2017-01-14 14:27发表

您好,
我在此处转载了您这篇博文, 感谢您的分享:
http://blog.csdn.net/lilai619/article/details/54426523

devwang_com 48楼 2017-01-06

xuexi~

Ascii_d 47楼 2016-12-07

<h></h>

LEandLA 46楼 2016-10-19

可以转转吗?

越停 45楼 2016-08-28 11:26发表

好棒, 清晰易懂!

---

dtcwyp 44楼 2016-08-17 15:40发表

赞一个

---

ipdome_lovelyfat 43楼 2016-08-07 23:21发表

不错, http://blog.csdn.net/liigo/article/details/582231

---

ipdome_lovelyfat 42楼 2016-08-07 22:53发表

http://blog.csdn.net/liigo/article/details/582231

=======
转载了

YehChiTian 41楼 2016-07-17 13:52发表

不错, 楼主我就默默的转走了。谢谢哈

bcbobo21cn 40楼 2016-07-13 13:30发表

你好;
我在此转载了你的博文;
http://blog.csdn.net/bcbobo21cn/article/details/51897211

wangzhione 39楼 2016-04-14 18:02发表

被标题吸引进来了, 楼主好文章. 干货. 要是楼主再写下篇, 关于gdb 内存堆栈, 多线程调试就更完美了.
楼主有机会, 也把你的博客也发到博客园中.

csdn广告太多了. 水经验也很多. 博客园中很干净, 更纯粹一些, 认真探讨的人也很多.

ligou8000 38楼 2016-03-17 13:52发表

通俗易懂 谢谢

AEsun 37楼 2016-03-03 13:09发表

太好了，收藏了！赞！

Joe3020 36楼 2015-12-28

您好。我在linux下跑MPI并行。r命令如何跑多个进程，比如 r -np 25 ./a.out?
另外，p如何专门指定比如说输出第三个进程的某个变量的值。
谢谢。

Dablelv 35楼 2015-12-10 1

简单易学，博主如果再出个gdb的进阶教程就好了！那就是锦上添花

wmb429006 34楼 2015-11-30 20:31发表

不错

攻城狮爱上程序猿 33楼 2015-11-12 19:51发表

很好的博客，谢谢博主。。。。。

flintlovesam 32楼 2015-11-04 20:43发表

支持一下学长

呀二呀 31楼 2015-10-26 17:14发表

很直观简洁，让人一目了然，楼主大赞

明镜亦非台-csdn 30楼 2015-09-15 17:58发表

感谢分享！拜读中……

liuxinglin 29楼 2015-09-09 21:19发表

太棒，不能赞更多！！

Fantasy1220 28楼 2015-07-31 16:01发表

主要用法加示例，很实用，赞

csdnligao 27楼 2015-07-22 23:26发表

太棒了～感谢～～～


诺尔曼 26楼 2015-07-21 17:30发表

可以转载?


vovo_ma 25楼 2015-07-20 08:53发表

很不错


u010227323 24楼 2015-06-15 17:26发表

发现博主跟我是校友!!


MGmoket 23楼 2015-06-02

清新可口,非常棒!


紫梦lan 22楼 2015-05-12 2

姗姗来迟,挺不错,学习中


任长江 21楼 2015-04-23 12:27发表

学习了,我该分析一下了。


shengyang998 20楼 2014-12-31 16:22发表

谢谢博主哦!


o-pqy-o 19楼 2014-11-08 17:16发表

写得太详细了!真诚请求转载,楼主~可以么。。。


ysh_ysh 18楼 2014-10-31 16:49发表

你好,看到你的这篇文章很不错,想转载保存,可以吗?


Liigo Re: 2014-11-03 17:21发表

回复ysh_ysh:欢迎转载,注明本文原始链接即可。


岁月小龙 17楼 2014-09-23 09:22发表

xie de taihaole


Open_Mind 16楼 2014-04-16 15:43发表

好文,谢谢。


wuyunpengleo 15楼 2013-11-08 19:47发表

赞一个

14楼 2013-08-24 23:50发表

非常感谢，这么一看之后马上就知道怎么进行初步的debug了。

工程师WWW

13楼 2013-08-23 12:25发表

“/n" 应该是"\n“，拷贝导致的错误？

Liigo

Re: 2013-08-29 20:27发表

回复工程师WWW：是提交CSDN博客时系统自动转换导致的。这种行为是很变态的，CSDN作为程序员的社区，犯这么严重的错误，难以置信。我多年前就已经严厉抨击过此事。

工程师WWW

Re: 2013-08-29

回复Liigo：原来如此啊，难怪经常遇到人家贴的代码编译不过的小错误。csdn还有bug，比如文档分类上下顺序调整的时候。

m_hook

12楼 2013-04-24 18:16发表

非常感谢（此处省略10000字）。对我非常有帮助。

zycfeixiang

11楼 2013-03-22 10:02发表

太感谢了,非常详细

loudyten

10楼 2013-03-09 19:28发表

受教了,楼主厉害，学习膜拜当中

HunDanXiaoLang

9楼 2013-01-18 22:52发表

刚找到你的博客就来说声谢谢了

HunDanXiaoLang

8楼 2013-01-18 22:51发表

感谢liigo老师编写的易语言零起点教程

shanhe1986

7楼 2012-12-04 11:04发表

原唱赞一下，很详细，感谢分享

星00恋

6楼 2012-07-25 14:24发表

不错，适合初学的我

The fool

5楼 2012-07-12 21:35发表

可能版本问题，在ubuntu11下测试时，例子程序中的换行符"/n",换成了"\n"才能正常换行。

玩好玩

Re: 2015-07-22 11:34发表

回复The fool:看14楼

GeoAnt 4楼 2011-10-17 00:31发表

好

hzlhzy 3楼 2011-03-13 01:50发表

姗姗来迟,挺不错,学习中

xiaoxu 2楼 2006-06-26

写得太棒了，
我就喜欢这种用实例来讲解内容的文章了.

farproc 1楼 2006-01-17 1[illegible]发表

不错,收藏了.

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场