

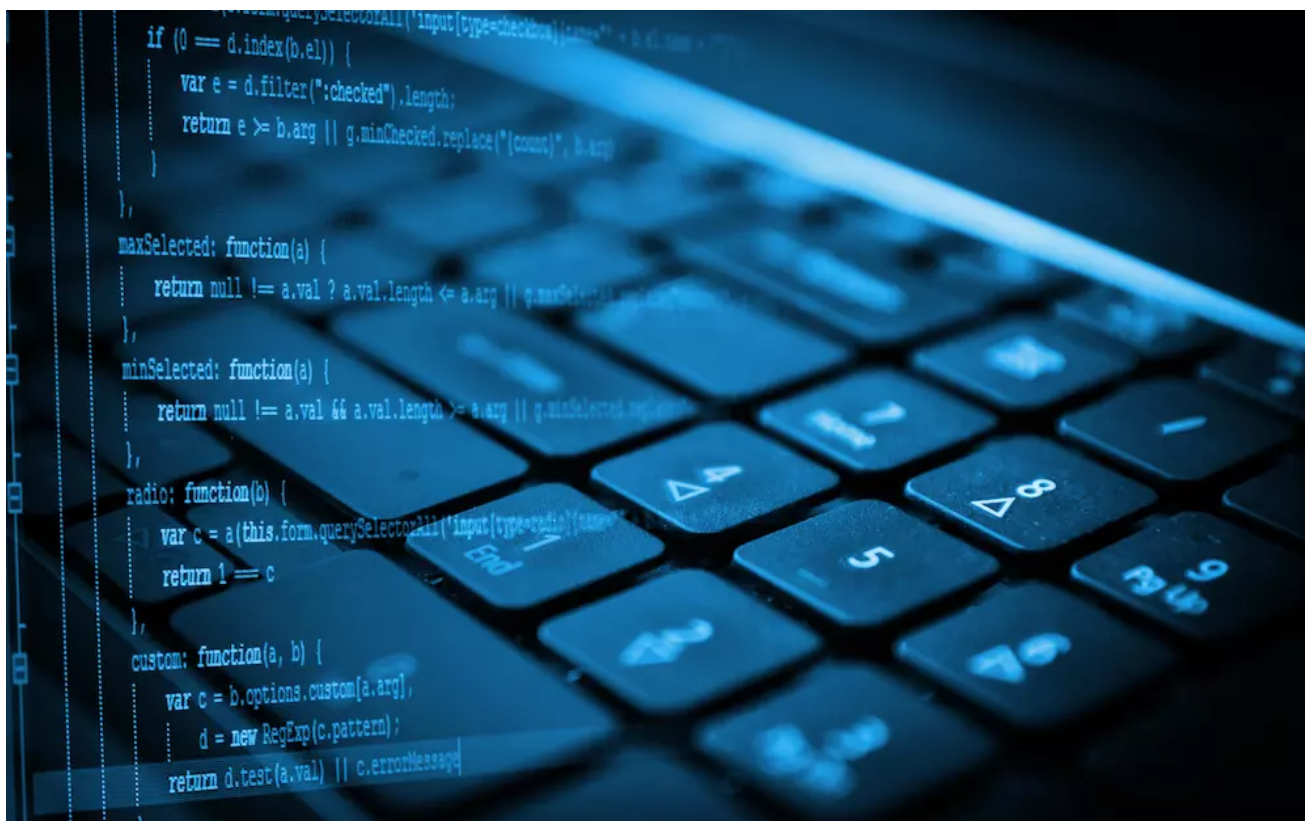
从 0 开始学习 Linux 系列之「08.15 个 gdb 调试基础命令」



程序小歌 (/u/b27c3fe5ed63) [+ 关注](#)

0.4 2017.07.20 18:03* 字数 2252 阅读 1842 评论 1 喜欢 11

(/u/b27c3fe5ed63)



代码键盘

版权声明：本文为 cdeveloper 原创文章，可以随意转载，但必须在明确位置注明出处！

gdb 简介

gdb 是 UNIX 及 UNIX-like 下的调试工具，在 Linux 下一般都直接在命令行中用 gdb 来调试程序，相比 Windows 上的集成开发环境 IDE 提供的图形界面调试，一开始使用 gdb 调试可能会让你感到生无可恋，但是只要熟悉了 gdb 调试的常用命令，调试出程序

会很有成就感，一方面因为这些命令就类似图形界面调试按钮背后的逻辑，另一方面用命令行来调试程序，逼格瞬间就上了一个档次，这次就**跟大家分享 gdb 调试的基本技术和 15 个常用调试命令**。

在此之前，我们先来回顾下在 Windows 上使用 IDE 的图形界面调试过程。

IDE 的调试步骤

在 Windows 的 IDE 下调试程序，例如使用 VS，一般都有下面这几个操作：

1. Debug 模式编译并启动程序
2. 程序运行出错，打**断点**分析出错的地方
3. 单步运行程序，包括：step over 单步执行；step into 跳入函数；step return 跳出函数
4. 还有全速运行，打印或者监视变量，冻结或解冻线程等调试技术

在 IDE 中上面的这些步骤一般都有**固定的按钮**提供给我们使用，非常的简单方便，我们只要多练习练习，在图形界面调试程序不会很难，但是在 Linux 下用命令来调试程序就比图形界面要复杂很多了。

其实，你知道**真正的调试高手**是什么样的吗？就是 Ta **对计算机原理和程序本身的逻辑理解非常深刻**，在 Ta 的脑海中已经可以模拟程序的运行过程，并且知道可能出错的地方，这样连断点都不用打了，而且 Bug 的命中率也不低，或许这就是真正的大佬吧 :)

我们回到正题，来介绍在 Linux 使用命令行的调试过程。

gdb 的调试步骤

在 Linux 下既然是使用命令行来调试，顾名思义就是**手敲命令来调试程序**，大体分为下面几个步骤，后面会详细介绍：

1. 编译可以调试的程序
2. 运行程序，打断点
3. 单步调试，监控变量
4. 可视化调试
5. 其他调试技术

可以看出，与 IDE 调试过程差不多，但是实际操作起来可是千差万别，可不是点按钮了，而是自己敲调试程序的命令，就相当于你正在学习那些调试按钮背后的原理，把这种方法学会，不用学就会使用 IDE 来调试程序，不管你信不信，我反正是信了 :)

下面开始正式的调试技术介绍。

15 个 gdb 调试基础命令

下面来正式介绍 gdb 常用的调试技术，都是调试命令，只看不做比较乏味，还是**建议你跟我一起动手调试下面的程序**，这样才能真正的学会，这是本次要调试的 `hello.c` 程序，非常简单：

```
#include <stdio.h>

int add(int x, int y) {
    return x + y;
}

int main() {
    int a = 1;
    int b = 2;
    printf("a = %d\n", a);
    printf("b = %d\n", b);

    int c = add(a, b);
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

1. 编译可以调试的程序

我们平常使用 gcc 编译的程序如果不加 `[-g]` 选项：

```
gcc hello.c -o hello
```

`gdb` 会提示该可执行文件**没有调试符号**，不能调试：

```
gdb hello
# gdb 提示信息
Reading symbols from a.out...(no debugging symbols found)...done.
```

如果需要在编译的时候加上 `[-g]` 参数：

```
gcc -g hello.c -o hello
```

2. 载入要调试的程序

我们在命令行下需要手动载入待调试的程序，有 2 种方法：

方法一 - gdb 可执行文件

使用如下的命令来载入可执行文件 `hello` 到 `gdb` 中：

```
gdb hello
```

载入成功，`gdb` 会打印一段提示信息，并且**命令行前缀变为** `(gdb)`，下面是我的 `Ubuntu` 打印的信息：

```
GNU gdb (Ubuntu 7.11.90.20161005-0ubuntu1) 7.11.90.20161005-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello...done.
(gdb) q
```

注：按 `q` 退出 `gdb`

方法二 - 使用 gdb 提供的 file 命令

第二种方法是在 `gdb` 环境中使用 `file` 命令，我们需要先进入 `gdb` 环境下：

```
gdb
```

使用 `file hello` 载入待调试程序：

```
...
(gdb) file hello
Reading symbols from hello...done.
(gdb) q
```

3. 查看调试程序

在 `gdb` 下查看调试程序使用命令 `list` 或简写 `l`，「回车」列出后面程序：

```
(gdb) list
1  #include <stdio.h>
2
3  int add(int x, int y) {
4      return x + y;
5  }
6
7
8  int main() {
9      int a = 1;
10     int b = 2;
(gdb)
11     printf("a = %d\n", a);
12     printf("b = %d\n", b);
13
14     int c = add(a, b);
15     printf("%d + %d = %d\n", a, b, c);
16     return 0;
17 }
```

4. 添加断点

在 gdb 下添加断点使用命令 `break` 或简写 `b`，有下面几个常见用法（这里统一用 `b`）：

1. `b function_name`
2. `b row_num`
3. `b file_name:row_num`
4. `b row_num if condition`

比如我们以第一个为例，在 `main` 函数上添加断点：

```
(gdb) b main
Breakpoint 1 at 0x6e8: file hello.c, line 4.
```

打印的信息告诉我们在 `hello.c` 文件的第 4 行，地址 `0x6e8` 处添加了一个断点，那如何查看断点呢？

5. 查看断点

在 gdb 下查看断点使用命令 `info break` 或简写 `i b`，比如查看刚才打的断点：

```
(gdb) i b
Num      Type           Disp Enb Address            What
1        breakpoint    keep y   0x00000000000006e8 in main at hello.c:4
```

可以看到打印出刚才添加的 `main` 函数的断点信息：编号，类型，显示状态，是否启用，地址，其他信息，那又如何删除这个断点呢？

6. 禁用断点

在 gdb 下禁用断点使用命令 `disable Num`，比如禁用刚才打的断点：

```
(gdb) disable 1
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep n   0x000000000000006e8 in main at hello.c:4
```

可以看到字段「Enb」已经变为 `n`，表示这个断点已经被禁用了。

7. 删除断点

在 gdb 下删除断点使用命令 `delete 断点 Num` 或简写 `d Num`，比如删除刚才的 `Num = 1` 的断点：

```
(gdb) d 1
(gdb) i b
No breakpoints or watchpoints.
```

删除后再次查看断点，提示当前没有断点，说明删除成功啦，下面来运行程序试试。

8. 运行程序

在 gdb 下使用命令 `run` 或简写 `r` 来运行当前载入的程序：

```
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello
a = 1
b = 2
1 + 2 = 3
[Inferior 1 (process 10415) exited normally]
```

我这次没有添加断点，程序全速运行，然后正常退出了。

9. 单步执行下一步

在 gdb 下使用命令 `next` 或简写 `n` 来单步执行下一步，假设我们在 `main` 打了断点：

```
(gdb) b main
Breakpoint 1 at 0x6e8: file hello.c, line 4.
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello

Breakpoint 1, main () at hello.c:4
4      int a = 1;
(gdb) n
5      printf("a = %d\n", a);
```

可以看到当前停在 `int a = 1;` 这一行，按 `n` 执行了下一句代码 `printf("a = %d\n", a);`

10. 跳入，跳出函数

在 gdb 下使用命令 `step` 或简写 `s` 来跳入一个函数，使用 `finish` 来跳出一个函数，我们在第 14 行 `int c = add(a, b);` 添加一个断点：

```
(gdb) b 14
Breakpoint 1 at 0x6f6: file hello.c, line 14.
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello
a = 1
b = 2

Breakpoint 1, main () at hello.c:14
14      int c = add(a, b);
(gdb) s
add (x=1, y=2) at hello.c:4
4      return x + y;
(gdb) finish
Run till exit from #0  add (x=1, y=2) at hello.c:4
0x000055555554705 in main () at hello.c:14
14      int c = add(a, b);
Value returned is $1 = 3
(gdb) n
15      printf("%d + %d = %d\n", a, b, c);
```

这个过程是这样的：

1. 在 14 行 `int c = add(a, b);` 添加断点
2. 程序运行并停到 `int c = add(a, b);` 这一行
3. `s` 跳入 `add` 函数
4. `finish` 跳出 `add` 函数，并输出一些函数返回的信息

11. 打印变量

在 gdb 中使用命令 `print var` 或简写 `p var` 来打印一个变量或者函数的返回值，我们在第 10 行 `int b = 2;` 添加一个断点：

```
(gdb) b 10
Breakpoint 1 at 0x6c3: file hello.c, line 10.
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello

Breakpoint 1, main () at hello.c:10
10      int b = 2;
(gdb) p a
$1 = 1
```

我们打印出变量 `a` 的值为 1，在调试中比较频繁的操作是「监视变量」，在 gdb 中如何做呢？

12. 监控变量

在 gdb 中使用命令 `watch var` 来监控一个变量，使用 `info watch` 来查看监控的变量，我们这里来监控变量 `c`：

```
(gdb) b 14
Breakpoint 1 at 0x6f6: file hello.c, line 14.
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello
a = 1
b = 2

Breakpoint 1, main () at hello.c:14
14      int c = add(a, b);
(gdb) watch c
Hardware watchpoint 2: c
(gdb) info watch


| Num | Type          | Disp | Enb | Address | What |
|-----|---------------|------|-----|---------|------|
| 2   | hw watchpoint | keep | y   |         | c    |


```

注意：程序必须要先运行才能监控。

13. 查看变量类型

在 gdb 下使用命令 `whatis` 查看一个变量的类型：

```
(gdb) b 10
Breakpoint 1 at 0x6c3: file hello.c, line 10.
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello

Breakpoint 1, main () at hello.c:10
10      int b = 2;
(gdb) whatis b
type = int
```


这里变量 `b` 是 `int` 类型。

14. 在 gdb 中进入 shell

在 gdb 下使用命令 `shell` 启动 shell：

```
(gdb) shell
orange@ubuntu:~/Desktop/gdb$ exit
exit
(gdb)
```

使用 `exit` 会再次退回到 gdb 中。

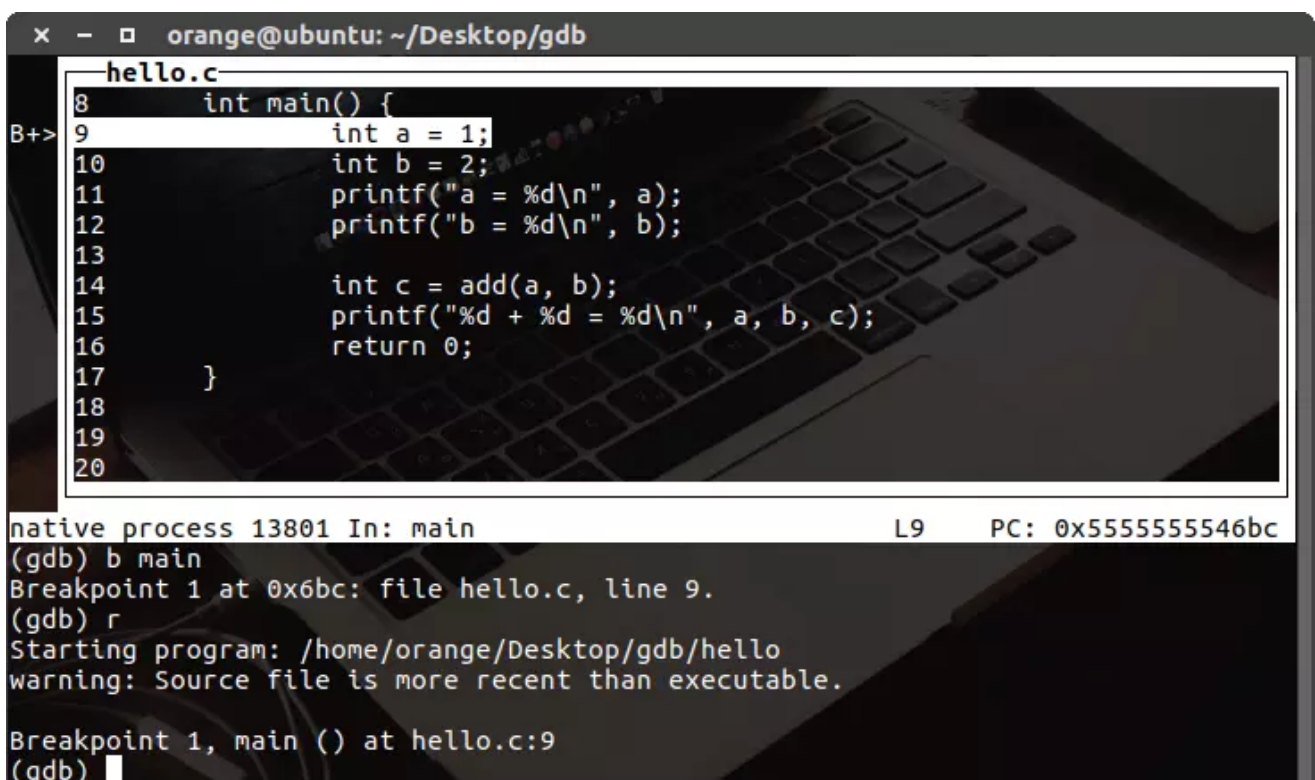
15. 在 gdb 中实现可视化调试

谁说 gdb 只能在命令行调试呢？gdb 也支持「图形界面」，不过这里的图形界面都是用字符显示的，当然不如 VS 那种好看，不过使用可视化相比直接看命令行更加直观了。

在 gdb 下使用 `wi` 启动可视化调试：

```
(gdb) wi
```

效果如下图所示，上面是代码效果，下面是命令界面：



```
orange@ubuntu: ~/Desktop/gdb
hello.c
8      int main() {
9      int a = 1;
10     int b = 2;
11     printf("a = %d\n", a);
12     printf("b = %d\n", b);
13
14     int c = add(a, b);
15     printf("%d + %d = %d\n", a, b, c);
16     return 0;
17 }
18
19
20
native process 13801 In: main L9 PC: 0x5555555546bc
(gdb) b main
Breakpoint 1 at 0x6bc: file hello.c, line 9.
(gdb) r
Starting program: /home/orange/Desktop/gdb/hello
warning: Source file is more recent than executable.
Breakpoint 1, main () at hello.c:9
(gdb)
```

gdb 图形界面

有了图形界面，就**再对照着图形界面将前面的命令再练习练习**，看看自己手敲的命令背后到底做了些什么，加深下影响。

结语

这篇博客主要介绍了 **gdb 基本的调试技术**，一篇文章不可能面面俱到，还有很多命令没有介绍，如果你有兴趣的话，这里还有一份比较好的 gdb 快速学习指南 (<https://link.jianshu.com?t=http%3A%2F%2Fbeej.us%2Fguide%2Fbggdb%2F>) 送给爱学习的你，不用客气，叫我雷锋就好。

最后，感谢你的阅读，我们下次再见 :)