常用命令

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: 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。	(gdb) b 8
b <函数名称>		(gdb) b main
d [编号]	d: Delete breakpoint的简写，删除指定编号的某个断点，或删除所有断点。断点编号从1开始递增。	(gdb) d
s	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；s 相当于其它调试器中的“Step Into (单步跟踪进入)”；（必须在GCC编译时使用“-g”参数）	(gdb) s
	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。n	

n	相当于其它调试器中的“Step Over (单步跟踪)”。（必须在GCC编译时使用“-g”参数）	(gdb) n
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 g_val
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
b *<函数名称>	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。	(gdb) b *main
b *<代码地址		(gdb) b *0x80483 5c

>		
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

具体调试例子

- 完整代码

```
//gdb-sample.c
#include <stdio.h>
int nGlobalVar = 0;

int tempFunction(int a, int b)
{
    printf("tempFunction is called, a = %d, b = %d /n", a, b);
    return (a + b);
}

int main(){
    int n;
    n = 1;
    n++;
    n--;

    nGlobalVar += 100;
    nGlobalVar -= 12;

    printf("n = %d, nGlobalVar = %d /n", n, nGlobalVar);

    n = tempFunction(1, 2);
    printf("n = %d", n);

    return 0;
}
```

- 编译命令

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

- 在该代码目录下输入命令"gdb"进入GDB
- “file”命令载入被调试程序 gdb-sample（编译生成的可执行文件）

```
(gdb)  file gdb-sample
```

> 此时输入命令“l”可以显示源代码

- 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/gdb-sample
n = 1, nGlobalVar = 88
tempFunction is called, a = 1, b = 2
n = 3
Program exited normally.
```

- 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.
```

> 再使用 r 命令执行调试文件，程序会停在设置的断点。打印如下：
> (gdb) r
> Starting program: /home/liigo/temp/gdb-sample
> Breakpoint 1, main () at gdb-sample.c:19 //断点
> 19 n = 1; //表示下一条要执行的源代码

- 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++;”。

- 用“p”命令（Print）看一下变量 n 的值

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

> 由于上句源代码已经执行了“n = 1"，所以n值为1;

- 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

```
(gdb) c
Continuing.
Breakpoint 2, main () at gdb-sample.c:26        //在第二个断点停下
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  //在第三个断点停下
12 printf("tempFunction is called, a = %d, b = %d /n", a, b);

(gdb) p a
$3 = 1

(gdb) p b
```

```
$4 = 2

(gdb) c
Continuing.
tempFunction is called, a = 1, b = 2
n = 3
Program exited normally. //程序正常运行结束
```

- 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)
```

- 命令“q”，退出（Quit）GDB调试环境：

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

参考链接

- 陈皓大佬 https://blog.csdn.net/haoel/article/category/9197
- https://blog.csdn.net/liigo/article/details/582231