# GDB 调试精粹及使用实例

一：列文件清单

1． List

(gdb) list line1,line2

二：执行程序

要想运行准备调试的程序，可使用 run 命令，在它后面可以跟随发给该程序的任何参数，包括标准输入和标准输出说明符(<和>)和外壳通配符（*、？、[、]）在内。

如果你使用不带参数的 run 命令，gdb 就再次使用你给予前一条 run 命令的参数，这是很有用的。

利用 set args 命令就可以修改发送给程序的参数，而使用 show args 命令就可以查看其缺省参数的列表。

（gdb）set args –b –x

(gdb) show args

backtrace 命令为堆栈提供向后跟踪功能。

Backtrace 命令产生一张列表，包含着从最近的过程开始的所以有效过程和调用这些过程的参数。

三：显示数据

利用 print 命令可以检查各个变量的值。

(gdb) print p (p 为变量名)

whatis 命令可以显示某个变量的类型

(gdb) whatis p

type = int *

print 是 gdb 的一个功能很强的命令，利用它可以显示被调试的语言中任何有效的表达式。表达式除了包含你程序中的变量外，还可以包含以下内容：

l 对程序中函数的调用

(gdb) print find_entry(1,0)

l 数据结构和其他复杂对象

(gdb) print *table_start

$8={e=reference=’\000’,location=0x0,next=0x0}

l 值的历史成分

(gdb)print $1 ($1 为历史记录变量,在以后可以直接引用 $1 的值)

l 人为数组

人为数组提供了一种去显示存储器块（数组节或动态分配的存储区）内容的方法。早期的调试程序没有很好的方法将任意的指针换成一个数组。就像对待参数一样，让我们查看内存中在变量 h 后面的 10 个整数，一个动态数组的语法如下所示：

base@length

因此，要想显示在 h 后面的 10 个元素，可以使用 h@10：

(gdb)print h@10

$13=(-1,345,23,-234,0,0,0,98,345,10)

四：断点(breakpoint)
break 命令（可以简写为 b）可以用来在调试的程序中设置断点，该命令有如下四种形式：
l break line-number 使程序恰好在执行给定行之前停止。
l break function-name 使程序恰好在进入指定的函数之前停止。
l break line-or-function if condition 如果 condition（条件）是真，程序到达指定行或函数时停止。
l break routine-name 在指定例程的入口处设置断点

如果该程序是由很多原文件构成的，你可以在各个原文件中设置断点，而不是在当前的原文件中设置断点，其方法如下：
(gdb) break filename:line-number
(gdb) break filename:function-name

要想设置一个条件断点，可以利用 break if 命令，如下所示：
(gdb) break line-or-function if expr
例：
(gdb) break 46 if testsize==100

从断点继续运行：countinue 命令
五．断点的管理

1． 显示当前 gdb 的断点信息：
(gdb) info break
他会以如下的形式显示所有的断点信息：
Num Type Disp Enb Address What
1 breakpoint keep y 0x000028bc in init_random at qsort2.c:155
2 breakpoint keep y 0x0000291c in init_organ at qsort2.c:168
(gdb)
2.删除指定的某个断点：
(gdb) delete breakpoint 1
该命令将会删除编号为 1 的断点，如果不带编号参数，将删除所有的断点
(gdb) delete breakpoint
3.禁止使用某个断点
(gdb) disable breakpoint 1
该命令将禁止断点 1,同时断点信息的 (Enb)域将变为 n
4．允许使用某个断点
(gdb) enable breakpoint 1
该命令将允许断点 1,同时断点信息的 (Enb)域将变为 y
5．清除原文件中某一代码行上的所有断点
(gdb)clean number
注：number 为原文件的某个代码行的行号
六．变量的检查和赋值
l whatis:识别数组或变量的类型
l ptype:比 whatis 的功能更强，他可以提供一个结构的定义

l set variable:将值赋予变量
l print 除了显示一个变量的值外，还可以用来赋值

七．单步执行
l next
不进入的单步执行
l step
进入的单步执行
如果已经进入了某函数，而想退出该函数返回到它的调用函数中，可使用命令 finish
八．函数的调用
l call name 调用和执行一个函数
(gdb) call gen_and_sork( 1234,1,0 )
(gdb) call printf(“abcd”)
$1=4
l finish 结束执行当前函数，显示其返回值（如果有的话）

九．机器语言工具
有一组专用的 gdb 变量可以用来检查和修改计算机的通用寄存器，gdb 提供了目前每一台计算机中实际使用的 4 个寄存器的标准名字：
l $pc ： 程序计数器
l $fp ： 帧指针（当前堆栈帧）
l $sp ： 栈指针
l $ps ： 处理器状态

十．信号
gdb 通常可以捕捉到发送给它的大多数信号，通过捕捉信号，它就可决定对于正在运行的进程要做些什么工作。例如，按 CTRL-C 将中断信号发送给 gdb，通常就会终止 gdb。但是你或许不想中断 gdb，真正的目的是要中断 gdb 正在运行的程序，因此，gdb 要抓住该信号并停止它正在运行的程序，这样就可以执行某些调试操作。

Handle 命令可控制信号的处理，他有两个参数，一个是信号名，另一个是接受到信号时该作什么。几种可能的参数是：
l nostop 接收到信号时，不要将它发送给程序，也不要停止程序。
l stop 接受到信号时停止程序的执行，从而允许程序调试；显示一条表示已接受到信号的消息（禁止使用消息除外）
l print 接受到信号时显示一条消息
l noprint 接受到信号时不要显示消息（而且隐含着不停止程序运行）
l pass 将信号发送给程序，从而允许你的程序去处理它、停止运行或采取别的动作。
l nopass 停止程序运行，但不要将信号发送给程序。
例如，假定你截获 SIGPIPE 信号，以防止正在调试的程序接受到该信号，而且只要该信号一到达，就要求该程序停止，并通知你。要完成这一任务，可利用如下命令：
(gdb) handle SIGPIPE stop print
请注意，UNIX 的信号名总是采用大写字母！你可以用信号编号替代信号名
如果你的程序要执行任何信号处理操作，就需要能够测试其信号处理程序，为此，就需要一

种能将信号发送给程序的简便方法，这就是 signal 命令的任务。该 命令的参数是一个数字或者一个名字，如 SIGINT。假定你的程序已将一个专用的 SIGINT（键盘输入，或 CTRL-C；信号 2）信号处理程序设置成采 取某个清理动作，要想测试该信号处理程序，你可以设置一个断点并使用如下命令：
（gdb） signal 2
continuing with signal SIGINT(2)
该程序继续执行，但是立即传输该信号，而且处理程序开始运行.

十一. 原文件的搜索
search text:该命令可显示在当前文件中包含 text 串的下一行。
Reverse-search text:该命令可以显示包含 text 的前一行。

十二.UNIX 接口
shell 命令可启动 UNIX 外壳，CTRL-D 退出外壳，返回到 gdb.

十三.命令的历史
为了允许使用历史命令，可使用 set history expansion on 命令
(gdb) set history expansion on

小结：常用的 gdb 命令
backtrace 显示程序中的当前位置和表示如何到达当前位置的栈跟踪（同义词：where）
breakpoint 在程序中设置一个断点
cd 改变当前工作目录
clear 删除刚才停止处的断点
commands 命中断点时，列出将要执行的命令
continue 从断点开始继续执行
delete 删除一个断点或监测点；也可与其他命令一起使用
display 程序停止时显示变量和表达时
down 下移栈帧，使得另一个函数成为当前函数
frame 选择下一条 continue 命令的帧
info 显示与该程序有关的各种信息
jump 在源程序中的另一点开始运行
kill 异常终止在 gdb 控制下运行的程序
list 列出相应于正在执行的程序的原文件内容
next 执行下一个源程序行，从而执行其整体中的一个函数
print 显示变量或表达式的值
pwd 显示当前工作目录
pype 显示一个数据结构（如一个结构或 C++类）的内容
quit 退出 gdb
reverse-search 在源文件中反向搜索正规表达式
run 执行该程序
search 在源文件中搜索正规表达式
set variable 给变量赋值
signal 将一个信号发送到正在运行的进程

step 执行下一个源程序行，必要时进入下一个函数
undisplay display 命令的反命令，不要显示表达式
until 结束当前循环
up 上移栈帧，使另一函数成为当前函数
watch 在程序中设置一个监测点（即数据断点）
whatis 显示变量或函数类型
**********

GNU 的调试器称为 gdb，该程序是一个交互式工具，工作在字符模式。在 X Window 系统中，有一个 gdb 的前端图形工具，称为 xxgdb。gdb 是功能强大的调试程序，可完成如下的调试任务：

* 设置断点；
* 监视程序变量的值；
* 程序的单步执行；
* 修改变量的值。

在可以使用 gdb 调试程序之前，必须使用 -g 选项编译源文件。可在 makefile 中如下定义 CFLAGS 变量：

CFLAGS = -g

运行 gdb 调试程序时通常使用如下的命令：

gdb progname

在 gdb 提示符处键入 help，将列出命令的分类，主要的分类有：

* aliases：命令别名
* breakpoints：断点定义；
* data：数据查看；
* files：指定并查看文件；
* internals：维护命令；
* running：程序执行；
* stack：调用栈查看；
* statu：状态查看；
* tracepoints：跟踪程序执行。

键入 help 后跟命令的分类名，可获得该类命令的详细清单。

gdb 的常用命令
命令 解释

break NUM 在指定的行上设置断点。

bt 显示所有的调用栈帧。该命令可用来显示函数的调用顺序。

clear 删除设置在特定源文件、特定行上的断点。其用法为 clear FILENAME:NUM

continue 继续执行正在调试的程序。该命令用在程序由于处理信号或断点而 导致停止运行时。

display EXPR 每次程序停止后显示表达式的值。表达式由程序定义的变量组成。

file FILE 装载指定的可执行文件进行调试。

help NAME 显示指定命令的帮助信息。

info break 显示当前断点清单，包括到达断点处的次数等。

info files 显示被调试文件的详细信息。
info func 显示所有的函数名称。
info local 显示当函数中的局部变量信息。
info prog 显示被调试程序的执行状态。
info var 显示所有的全局和静态变量名称。
kill 终止正被调试的程序。
list 显示源代码段。
make 在不退出 gdb 的情况下运行 make 工具。
next 在不单步执行进入其他函数的情况下，向前执行一行源代码。
print EXPR 显示表达式 EXPR 的值。

******gdb 使用范例**********

---

清单 一个有错误的 C 源程序 bugging.c
代码:

---

```
1   #i nclude
2
3   static char buff [256];
4   static char* string;
5   int main ()
6   {
7         printf ("Please input a string: ");
8         gets (string);
9       printf ("\nYour string is: %s\n", string);
10 }
```

---

上面这个程序非常简单，其目的是接受用户的输入，然后将用户的输入打印出来。该程序使用了一个未经过初始化的字符串地址 string，因此，编译并运行之后，将出现 Segment Fault 错误：

```
$ gcc -o bugging -g bugging.c
$ ./bugging
Please input a string: asfd
Segmentation fault (core dumped)
```

为了查找该程序中出现的问题，我们利用 gdb，并按如下的步骤进行：
1．运行 gdb bugging 命令，装入 bugging 可执行文件；
2．执行装入的 bugging 命令 run；
3．使用 where 命令查看程序出错的地方；
4．利用 list 命令查看调用 gets 函数附近的代码；
5．唯一能够导致 gets 函数出错的因素就是变量 string。用 print 命令查看 string 的值；
6．在 gdb 中，我们可以直接修改变量的值，只要将 string 取一个合法的指针值就可以了，

为此，我们在第 8 行处设置断点 break 8；
7．程序重新运行到第 8 行处停止，这时，我们可以用 set variable 命令修改 string 的取值；
8．然后继续运行，将看到正确的程序运行结果。