1. **GCC（GNU编译器套件）**

GNU编译器套件（GNU Compiler Collection）包括[C](https://baike.baidu.com/item/C/7252092" \t "_blank)、[C++](https://baike.baidu.com/item/C%2B%2B" \t "_blank)、[Objective-C](https://baike.baidu.com/item/Objective-C" \t "_blank)、[Fortran](https://baike.baidu.com/item/Fortran" \t "_blank)、[Java](https://baike.baidu.com/item/Java/85979" \t "_blank)、[Ada](https://baike.baidu.com/item/Ada/5606819" \t "_blank)和[Go语言](https://baike.baidu.com/item/Go%E8%AF%AD%E8%A8%80" \t "_blank)的前端，也包括了这些语言的库（如libstdc++、libgcj等等）。虽然我们称GCC是[C语言](https://baike.baidu.com/item/C%E8%AF%AD%E8%A8%80" \t "_blank)的[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8)，但使用gcc由C语言[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81" \t "_blank)文件生成[可执行文件](https://baike.baidu.com/item/%E5%8F%AF%E6%89%A7%E8%A1%8C%E6%96%87%E4%BB%B6)的过程不仅仅是编译的过程，而是要经历四个相互关联的步骤∶[预处理](https://baike.baidu.com/item/%E9%A2%84%E5%A4%84%E7%90%86)(也称[预编译](https://baike.baidu.com/item/%E9%A2%84%E7%BC%96%E8%AF%91" \t "_blank)，Preprocessing)、[编译](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91" \t "_blank)(Compilation)、[汇编](https://baike.baidu.com/item/%E6%B1%87%E7%BC%96" \t "_blank)(Assembly)和链接(Linking)。

**基本用法**

在使用GCC[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8)的时候，我们必须给出一系列必要的调用参数和文件名称。GCC编译器的调用参数大约有100多个，这里只介绍其中最基本、最常用的参数。具体可参考GCC Manual。

**GCC最基本的用法是∶gcc [options] [filenames]**

**其中options就是编译器所需要的参数，filenames给出相关的文件名称。**

-c，只[编译](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91" \t "_blank)，不链接成为[可执行文件](https://baike.baidu.com/item/%E5%8F%AF%E6%89%A7%E8%A1%8C%E6%96%87%E4%BB%B6)，编译器只是由输入的.c等[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81" \t "_blank)文件生成.o为后缀的目标文件，通常用于编译不包含主程序的[子程序](https://baike.baidu.com/item/%E5%AD%90%E7%A8%8B%E5%BA%8F" \t "_blank)文件。

**-o output\_filename，**确定输出文件的名称为output\_filename，同时这个名称不能和源文件同名。如果不给出这个选项，gcc就给出预设的可执行文件a.out。

-g，产生符号调试工具(GNU的gdb)所必要的符号资讯，要想对源代码进行调试，我们就必须加入这个选项。

-O，对程序进行优化[编译](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91" \t "_blank)、链接，采用这个选项，整个[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81)会在编译、链接过程中进行优化处理，这样产生的[可执行文件](https://baike.baidu.com/item/%E5%8F%AF%E6%89%A7%E8%A1%8C%E6%96%87%E4%BB%B6)的执行效率可以提高，但是，编译、链接的速度就相应地要慢一些。

-O2，比-O更好的优化编译、链接，当然整个编译、链接过程会更慢。

-Idirname，将dirname所指出的目录加入到程序头文件目录列表中，是在[预编译](https://baike.baidu.com/item/%E9%A2%84%E7%BC%96%E8%AF%91" \t "_blank)过程中使用的参数。C程序中的头文件包含两种情况∶

A)#include <myinc.h>

B)#include “myinc.h”

其中，A类使用尖括号(< >)，B类使用双引号(“ ”)。对于A类，[预处理](https://baike.baidu.com/item/%E9%A2%84%E5%A4%84%E7%90%86" \t "_blank)程序cpp在系统预设包含[文件目录](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E7%9B%AE%E5%BD%95" \t "_blank)(如/usr/include)中搜寻相应的文件，而B类，预处理程序在[目标文件](https://baike.baidu.com/item/%E7%9B%AE%E6%A0%87%E6%96%87%E4%BB%B6" \t "_blank)的文件夹内搜索相应文件。

**二、Gdb的用法**

  gdb调试器时一款GNU组织开发、发布的UNIX/Linux环境下的程序调试工具，没有图形界面，但功能强大。

**（一）GDB使用流程：**

 保存后用gcc对文件进行编译，需要加上 -g 参数，否则不能使用gdb进行调试。

**gcc gdbTest.c -o gdbTest –g**

1. 查看文件

       在gdb中键入“l”(list)就能够查看载入的文件。

在不输入其他参数的情况下l 打开的是包含main()函数的文件，如果要打开其他文件可以在l 后面输入文件名作为参数。

       也可以定位到某个文件下的某个函数的位置，使用的方式是在文件名后面再附加一个函数名字符串，例如：

**l gdbTest.c:sum**

1. 设置断点

       设置断点的方式是输入b(或者 break) 行号，例如再gdbTest.c中的第6行和第21行分别设置一个断点，我们可以：

       b 6

       b 21

除了普通断点以外，还可以设置条件断点：

              b 22 if i==10

1. 删除断点

       使用命令 d + 断点号 可以删除指定的断点

1. 查看断点

       在设置完断点后还可以查看断点信息：

              info b

1. 运行代码

       在gdb中运行代码可以使用 start 命令。

运行过程中遇到断点就会停下来。

       继续执行输入命令             c(Continue)

       执行下一步输入命令          n(Next)

       继续执行输入命令             r(Run)

       进入函数体输入命令          s (Step)

       跳出循环使用命令             u

       跳出当前函数使用命令      finish

1. 查看变量值

       查看变量的值可以使用命令p + 变量名

       p i

1. 追踪变量的值可以使用命令 display + 变量名

       display i

1. info display 命令可以查看到追踪的变量的信息
2. 使用命令 undisplay + 变量名可以停止追踪某个变量
3. 查看变量类型

       使用命令type + 变量名 可以查看变量的类型

1. 设置变量的值

       使用命令 set var + 变量名 = 值 可以直接设置某个变量的值。

1. 退出gdb

       使用命令 quit 可以退出gdb

**GDB基本命令：**

       gdb的命令可以通过help进行查找，因为gdb的命令非常多，所以gdb的help将其分成了很多种类（class），用户可以通过进一步查看相关的class找到相应的命令。

工作环境相关命令

       set args[运行时参数]

              指定运行时参数，如：set args 2

       show args

              查看设置好的运行参数

       path dir

              设定程序的运行路径

       show paths

              查看程序的运行路径

       set environment var[=value]

              设置环境变量

       show environment [var]

              查看环境变量

       cd dir

              进入dir目录，相当于shell中的cd命令

       pwd

              显示当前工作目录

       shell command

              运行shell的command命令

**设置断点于恢复命令**

       info

              查看所设断点

       break[文件名:]行号或函数名 <条件表达式>

              设置断点

       tbreak[文件名:]行号或函数名 <条件表达式>

              设置临时断点，到达后被自动删除

       delete [断点号]

              删除断点

       disable [断点号]

              停止指定断点，不等同于删除，可以被恢复

enable [断点号]

              激活指定断点，被disable的断点

       condition [断点号] <条件表达式>

              修改对应断点条件

       ignore [断点号] num

              忽略断点num次

       step

              单步恢复程序，并进入函数调用

       neat

              单步恢复程序，不进入函数调用

       finish

              运行程序，直到当前函数完成返回

       c

              继续执行，直到函数结束或遇到新断点

**gdb中源码查看相关命令**

       list <行号>|<函数名>

              查看指定位置代码

       file [文件名]

              加载指定文件

       forward-search 正则表达式

              源代码的前向搜索

       reverse-search 正则表达式

              源代码的后向搜索

       dir DIR

              将路径DIR添加到源文件搜索路径的开头

       show directories

              显示源文件的当前搜索路径

       info line

              显示加载到gdb内存中的代码

**gdb中查看运行数据相关命令**

       print 表达式|变量

              查看程序运行时对应表达式或变量的值

       x <n/f/u>

              查看内存变量内容，n为整数表示显示内存长度，f表示显示的格式，u表示从当前地址往后请求显示的字符数

       display 表达式

              设定单步运行或其他情况中，自动显示的对应的表达式的内容

       backtrace

              查看当前栈的情况，既可以查到哪些被调用的函数尚未返回

**三、System函数的原理**

**system（执行shell 命令）**

相关函数

      fork，execve，waitpid，popen

表头文件

      #include<stdlib.h>

定义函数

      int system(const char \* string);

函数说明

**system()会调用fork()产生子进程，由子进程来调用/bin/sh-c string来执行参数string字符串所代表的命令，此命令执行完后随即返回原调用的进程。**在调用system()期间SIGCHLD 信号会被暂时搁置，SIGINT和SIGQUIT 信号则会被忽略。

返回值

      如果system()在调用/bin/sh时失败则返回127，其他失败原因返回-1。若参数string为空指针(NULL)，则返回非零值。如果 system()调用成功则最后会返回执行shell命令后的返回值，但是此返回值也有可能为system()调用/bin/sh失败所返回的127，因此最好能再检查errno 来确认执行成功。

附加说明

      在编写具有SUID/SGID权限的程序时请勿使用system()，system()会继承环境变量，通过环境变量可能会造成系统安全的问题。

system的实现

int system(const char \* cmdstring)

{

pid\_t pid;

int status;

if(cmdstring == NULL){

return (1);

}

if((pid = fork())<0){

status = -1;

}

else if(pid == 0){

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

-exit(127); //子进程正常执行则不会执行此语句

}

else{

while(waitpid(pid, &status, 0) < 0){

if(errno != EINTER){

status = -1;

break;

}

}

}

return status;

}

2.WIFEXITED(status) 这个宏用来指出子进程是否为正常退出的，如果是，它会返回一个非零值。

WEXITSTATUS(status) 当WIFEXITED返回非零值时，我们可以用这个宏来提取子进程的返回值，如果子进程调用exit(5)退出，WEXITSTATUS(status)就会返回5；如果子进程调用exit(7)，WEXITSTATUS(status)就会返回7。请注意，如果进程不是正常退出的，也就是说，WIFEXITED返回0，这个值就毫无意义。

判断一个system函数调用shell脚本是否正常结束的方法应该是如下3个条件同时成立：

（1）-1 != status

（2）WIFEXITED(status)为真

（3）0 == WEXITSTATUS(status)