**1)查看宏**

默认情况下，在GDB中是不能查看宏的值及定义的，但通过如下方法，则可以达到目的：

编译源代码时，加上“-g3-gdwarf-2”选项，请注意不是“-g”，必须为“-g3”，查看宏的值使用命令p，这和查看变量的值的方法相同，如果想查看宏的定义，使用 “-macroexpand”命令即可

**2) 如果执行一连串命令？**

这个借助命令“source”即可，“source”的参数为一个存有一串命令的文件名

**3) 如何同时给多个函数打断点？**

使用“rb”命令，如果执行“rb”时不带参数，则表示在所有函数处打一个断点，“rb”后面可以接一个符合正则表达式的参数，用来对符合正则表达式的所有函数打断点

**4) .gdbinit文件**

GDB在启动时，会在用户主目录中寻找这个文件，并执行该文件中所有命令，文件格式为：

define command-alias

command

end

如要给“b main”取一个别名“bm”：

define bm

b main

end

此外，还可以给这个别名加上帮助说明性文字，格式为：

document bm

帮助说明性文字

end

如，给“bm”别名添加帮助说明性文字：

document bm

break at main() function

end

这样，在使用“helpbm”时，GDB就会打印出“break at main()function”

**5) 自定义命令**

除了在.gdbinit文件自定义命令外，还可以直接在GDB运行环境定义命令，语法相同

**6) 定义命令钩子**

钩子用来在执行某个命令前或命令后，先执行某个或某些命令。假如想在print命令前显示一段 “----------”，则：

define hook-print

echo ----------/n

end

注意“hook-”后接的必须是命令全称，不能是缩写。  
如果想在命令执行完，再执行某个或某些命令，则：

*#define hookpost-print*

echo *----------/n*

end

**7) GDB中循环**

GDB支持if/else/while/loop\_break/loop\_continue，如：

(gdb) set $x=1

(gdb) while $x==1

>p $x

>set $x=0

>p $x

>end

-----------------

b$8 = 1

-----------------

b$9 = 0

**8) 函数测试**

这在UT中使用最多，同时也是部分ST用例的入口。GDB中可以在Ctrl+C暂停程序执行后，使用call命令调用程序中的任何函数，使用方法与被测程序语言的语法一致，如：

call foo(5, “Hello!”)

函数执行的上下文比较特殊，栈空间是GDB临时分配的特殊内存区域，进程空间与被测函数相同，但与任何具体线程无关。如果使用call进行UT，则 你会发现GDB的断点设置对call所调用的函数无效，这是GDB的一个默认保护：任何命令涉及的表达式（函数调用同样被视作表达式）都不受断点影响。为 了顺利进行UT，我们必须手动修改这一默认设置：

set debug expression 1

这样就可以break在被call所调用的函数中了。

Tip：如果函数中使用了类printf的输出函数，那么可能在call之后看不到任何输出，那是由于标准输出的缓冲造成的，只需简单的c(continue)一次就可以观察到输出了。

**9)简单的函数桩**

UT中对函数进行打桩是一件非常麻烦的事情，对于小项目，往往会浪费大量的精力。其实，通过GDB，我们可以变相的实现简单的函数桩，且较为便捷。  
基本思路就是在需要打桩的函数上设置断点，待进入函数后，根据对函数的打桩需求，可以分别采取如下措施：

（1）直接返回：用GDB的return命令强制返回即可，如果有返回值，直接跟在return后面即可，如：

return 5

（2）执行函数的部分逻辑：这需要借助GDB的执行路径篡改功能。  
“jump <行号>”直接跳至对应的代码行继续执行，也可以用“jump \*<地址>”跳转到机器指令地址。

Tip：在x86开发环境下，即使像VC这种不支持“执行路径篡改”功能的调试器，也可以通过直接修改寄存器PC的值达到跳转的目的

**10)临时数据的构造**

在测试一些包含复杂数据类型的代码时，常常需要构造一些复杂结构的临时数据，通常简单易行的办法是直接分配一块临时内存，如：（假设SOME\_TYPE是某种复杂数据类型，如typedef的struct、union、enum等）

set $temp = ({SOME\_TYPE \*}malloc(sizeof(SOME\_TYPE)))

然后为临时变量$temp填充相应的数据：

set $temp->some\_field = 7

最后将其用作普通的数据参与测试：（假设foo()函数接收SOME\_TYPE\*类型的参数）

call foo($temp)