基本用法

list(简写 l)

```
(gdb)l      # 默认打印当前行前后各4行
(gdb)l n    # 从第n行开始，打印十行
(gdb)l +n   # 从当前行的后n行开始，打印10行
(gdb)l -n   # 从当前行的前n行开始，向前打印10行
```

break(简写 b)

```
(gdb)b filename:func
(gdb)b filename:line    #如果是当前文件，则不需要指定文件名
(gdb)b 21 if var==0
# 条件断点也可分两步走，比如，已经设置了断点b 10，且断点编号为2，则可:
(gdb)condition 2 if var > 1
```

观察点

```
(gdb)watch expr     # 当表达式的值发生变化时，程序会停下来，并打印变化的信息
(gdb)watch var      # 当变量的值发生变换时，停住
(gdb)rwatch var     # 当变量被读时，停住
(gdb)awatch var     # 当变量被读或被写时，停住
```

> 没有 wwatch

捕捉点

捕捉程序运行时发生的一些事件，比如C++异常或加载共享库。[捕捉点]

```
(gdb)catch Event
```

维护停止点（断点、观察点、捕捉点）

```
(gdb)delete     # 删除所有的已设置的停止点
(gdb)clear      # 清除当前行的断点，也可以具体指定，如：clear func, clear line...
(gdb)disable    # 废除某停止点
(gdb)enable     # 新激活被废除的停止点
```

单步调试

```
(gdb)next       # 简写n。不进入函数体
(gdb)n i        # 单步执行i次
(gdb)step       # 简写s。进入函数体
(gdb)s i        # 单步执行i次
```

info

```
(gdb)info frame      # 显示当前栈帧的详细信息（栈帧的地址、寄存器的值、调用当前函数的地址、开发语言、形参及局部变量在栈中的位置等）
(gdb)info locals     # 显示局部变量，（注意，当某个命令想不起来了，记住首字母，然后按Tab键）
(gdb)info breakpoints        # 显示断点信息
(gdb)info watchpoints        # 显示观察点信息，实际上可能与上一个命令重复
```

examine(缩写 x)

```
(gdb)x/[number][format][b|h|w|g] <addr>      # 查看内存地址addr的值
```

number： 一个正整数，表示从当前地址向后显示几个地址单元的内容。如 `x/24 0x400c90` 表示查看 0x400c90 到向后 0x400c90+24 的内容

format：显示的格式不是查看的格式。和c语言中的格式缩写一样：d：整数；s：字符串；c：字符；u：无符号整数；o：八进制；x：十六进制；f：浮点数

[b|h|w|g]：地址单元大小。b(byte)、h(halfword)、w(word)、g(giant) 分别表示单、双、四、八字节。默认是w。

打印

```
(gdb)print var           # 简写p。打印变量var
(gdb)p arr[0]@length     # 打印数组
(gdb)p/x var             # 按十六进制格式输出. x/a:16进制；c:字符格式；u:无符号整形；o:八进制；t:二进制；f:浮点型
(gdb)display varlist     # 当程序停住时，自动显示varlist中的变量
(gdb)display/i  varlist      #输出格式为汇编
```

其他

```
~$gdb --args ./testprg arg1 arg2         # 调试带参数程序
(gdb)r arg1 arg2     # 把arg1 arg2作为程序的参数传入
(gdb)set args -x -i input -o output      # 传递参数 "-x -i input -o output"
(gdb)r < input  # 可以重定向标准输入或输出
(gdb)backtrace  # 简写 bt。显示堆栈信息（函数调用关系）
(gdb)until      # 简写u。退出循环体
(gdb)finish     # 退出当前函数。在main函数（最外层）中没意义
(gdb)shell cmd  # 执行shell 命令。cmd 可以是shell环境下的任意命令
(gdb)shell      # 暂时切出gdb环境，进入shell环境。可以使用exit切回gdb,继续之前的调试


```

调试正在运行的程序

```
~$gdb ./test pid
# 或者：
~$gdb
(gdb)attach pid      # 加载
...                  # 正常的调试操作
(gdb)detach          # 卸载
```

调试core文件

当出现内存非法访问时，一般会dump进程的镜像到core文件。

```
~$gdb ./test core
(gdb)where      # 将会打印当前的堆栈信息，以及出问题的行
```

调试多线程

```
(gdb)info threads       # 显示当前可调试的线程
(gdb)thread ID          # 切换到指定ID的线程
(gdb)break test.c:11 thread all         # 所有线程设置断点
(gdb)thread apply ID1 ID2 command       # 让一个或者多个线程执行命令command
```

在使用 `step` 或者 `continue` 命令调试某个线程的时候，其他线程也是同时执行的，怎么只让被调试线程执行呢？通过 `set scheduler-locking off|on|step` 可以实现。

- off：不锁定任何线程，当被调试线程执行时，其它线程也执行，默认情况。
- on：锁定其他线程，只有当前线程执行。
- step：在step（单步）时，只有被调试线程运行

多进程和多线程调试详见：gdb调试多进程与多线程*snow_5288*的博客-*CSDN*博客

参考

GDB的基本用法 - 逆转的旋律 - 博客园 (cnblogs.com)

GDB的进阶使用 - 逆转的旋律 - 博客园 (cnblogs.com)