Linux 常用命令：开发调试篇

守望先生 Linux爱好者 今天

（给Linux爱好者加星标，提升Linux技能）

> 作者：守望（本文来自作者投稿，简介见末尾）

前言

Linux常用命令中有一些命令可以在开发或调试过程中起到很好的帮助作用，有些可以帮助了解或优化我们的程序，有些可以帮我们定位疑难问题。本文将简单介绍一下这些命令。

示例程序

我们用一个小程序，来帮助后面我们对这些命令的描述，程序清单cmdTest.c如下:

```
#include<stdio.h>
int test(int a,int b)
{
    return a/b;
}
int main(int argc,char *argv[])
{
    int a = 10;
    int b = 0;
    printf("a=%d,b=%d\n",a,b);
    test(a,b);
    return 0;
}
```

编译获得elf文件cmdTest并运行：

```
gcc -g -o cmdTest cmdTest.c
./cmdTest
a=10,b=0
Floating point exception (core dumped)
```

程序内容是在main函数中调用test，计算a/b的值，其中b的值为0，因此程序由于除0错误异常终止。

查看文件基本信息--file

```
file cmdTest
cmdTest: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linke
```

通过file命令可以看到cmdTest的类型为elf，是64位、运行于x86-64的程序，not striped表明elf文件中还保留着符号信息以及调试信息等不影响程序运行的内容。

查看程序依赖库--ldd

```
ldd cmdTest
        linux-vdso.so.1 =>  (0x00007ffc8e548000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f0621931000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f0621cf6000)
```

我们可以看到cmdTest依赖了libc.so等库。

查看函数或者全局变量是否存在于elf文件中--nm

nm命令用于查看elf文件的符号信息。文件编译出来之后，我们可能不知道新增加的函数或者全局变量是否已经成功编译进去。这时候，我们可以使用nm命令来查看。
例如，查看前面所提到的elf文件有没有test函数，可以用命令：

```
nm cmdTest|grep test
000000000040052d T test   #打印结果
```

按照地址顺序列出符号信息：

```
nm -n cmdTest
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
                 w _Jv_RegisterClasses
                 w __gmon_start__
                 U __libc_start_main@@GLIBC_2.2.5
                 U printf@@GLIBC_2.2.5
00000000004003e0 T _init
0000000000400440 T _start
0000000000400470 t deregister_tm_clones
00000000004004a0 t register_tm_clones
00000000004004e0 t __do_global_dtors_aux
0000000000400500 t frame_dummy
000000000040052d T test
0000000000400540 T main
0000000000400590 T __libc_csu_init
```

```
0000000000400600 T __libc_csu_fini
（列出部分内容）
```

可以看到test函数的开始地址为0x000000000040052d，结束地址为0x0000000000400540。

打印elf文件中的可打印字符串--strings

例如你在代码中存储了一个版本号信息，那么即使编译成elf文件后，仍然可以通过strings搜索其中的字符串甚至可以搜索某个.c文件是否编译在其中：

```
strings elfFile| grep "someString"
```

查看文件段大小--size

可以通过size命令查看各段大小：

```
size cmdTest
   text       data       bss       dec       hex   filename
   1319       560        8      1887       75f    cmdTest
```

text段：正文段字节数大小

data段:包含静态变量和已经初始化的全局变量的数据段字节数大小

bss段：存放程序中未初始化的全局变量的字节数大小

当我们知道各个段的大小之后，如果有减小程序大小的需求，就可以有针对性的对elf文件进行优化处理。

为elf文件”瘦身“--strip

strip用于去掉elf文件中所有的符号信息：

```
ls -al cmdTest
-rwxr-xr-x 1 hyb root 9792 Sep 25 20:30 cmdTest #总大小为9792字节
strip cmdTest
ls -al cmdTest
-rwxr-xr-x 1 hyb root 6248 Sep 25 20:35 cmdTest#strip之后大小为6248字节
```

可以看到，“瘦身”之后，大小减少将近三分之一。但是要特别注意的是，“瘦身”之后的elf文件由于没有了符号信息，许多调试命令将无法正常使用，出现core dump时，问题也较难定位，因此只建议在正式发布时对其进行“瘦身”。

查看elf文件信息--readelf

readelf用于查看elf文件信息，它可以查看各段信息，符号信息等，下面的例子是查看elf文件头信息：

```
readelf -h cmdTest
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  #elf文件魔数字
  Class:                             ELF64  #64位 elf文件
  Data:                              2's complement, little endian#字节序为小端序
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V #
  ABI Version:                       0
  Type:                              EXEC (Executable file)#目标文件类型
  Machine:                           Advanced Micro Devices X86-64 #目标处理器体系
  Version:                           0x1
  Entry point address:               0x400440  #入口地址
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4456 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         28
  Section header string table index: 27
```

从elf头信息中，我们可以知道该elf是64位可执行文件，运行在x86-64中，且字节序为小端序。另外，我们还注意到它的入口地址是0x400440(_start)，而不是400540(main)。也就是说，我们的程序运行并非从main开始。

反汇编指定函数--objdump

objdump用于展示elf文件信息，功能较多，在此不逐一介绍。有时候我们需要反汇编来定位一些问题，可以使用命令：

```
objdump -d cmdTest #反汇编整个cmdTest程序
```

但是如果程序较大，那么反汇编时间将会变长，而且反汇编文件也会很大。如果我们已经知道了问题在某个函数，只想反汇编某一个函数，怎么处理呢？

我们可以利用前面介绍的nm命令获取到函数test的地址，然后使用下面的方式反汇编：

```
objdump -d cmdTest --start-address=0x40052d --stop-address=0x400540 ##反汇编指定地
```

端口占用情况查看--netstat

我们可能常常会遇到进程第一次启动后，再次启动会出现端口绑定失败的问题，我们可以通过netstat命令查看端口占用情况：

```
netstat -anp|grep 端口号
```

core dump文件生成配置--ulimit -c

有时候我们的程序core dump了却没有生成core文件，很可能是我们设置的问题：

```
ulimit -c #查看core文件配置，如果结果为0，程序core dump时将不会生成core文件
ulimit -c unlimited #不限制core文件生成大小
ulimit -c 10 #设置最大生成大小为10kb
```

调试神器--gdb

gdb是一个强大的调试工具，但这里仅介绍两个简单使用示例。

有时候程序可能已经正在运行，但是又不能终止它，这时候仍然可以使用gdb调试正在运行的进程：

```
gdb processFile PID #processFile为进程文件，pid为进程id，可通过ps命令查找到
```

有时候程序可能core dump了，但是系统还留给了我们一个礼物--core文件。

在core文件生成配置完成之后，运行cmdTest程序，产生core文件。我们可以用下面的方法通过core文件定位出错位置：

```
gdb cmdTest core #processFile为进程文件，core为生成的core文件
Core was generated by `./cmdTest'.
Program terminated with signal SIGFPE, Arithmetic exception.
#0  0x00000000004004fb in test (a=10, b=0) at cmdTest.c:4
4           return a/b;
(gdb)bt
#0  0x00000000004004fb in test (a=10, b=0) at cmdTest.c:4
```

```
#1  0x000000000040052c in main (argc=1, argv=0x7ffca9536d38) at cmdTest.c:10
(gdb)
```

输入bt后，就可以看到调用栈了,出错位置在test函数，cmdTest.c的第4行。

定位crash问题--addr2line

有时候程序崩溃了但不幸没有生成core文件，是不是就完全没有办法了呢？还是cmdTest的例子。运行完cmdTest之后，我们通过dmesg命令可以获取到以下内容

```
[27153070.538380] traps: cmdTest[2836] trap divide error ip:40053b sp:7ffc230d92
```

该信息记录了cmdTest运行出错的基本原因（divide error）和出错位置（40053b）,我们使用addr2line命令获取出错具体行号：

```
addr2line -e cmdTest 40053b
/home/hyb/practice/cmdTest.c:4
```

可以看到addr2line命令将地址(40053b)翻译成了文件名(cmdTest.c)和行号(4)，确定了出错位置。

总结

本文对以上命令仅介绍其经典使用，这些命令都还有其他一些有帮助的用法，但由于篇幅有限，不在此介绍，更多使用方法可以通过man 命令名的方式去了解。

【本文作者】

守望：一名好文学，好技术的开发者。在个人公众号“编程珠玑”坚持分享原创技术文章，期待一起交流学习。

推荐阅读 （点击标题可跳转阅读）

Linux 常用命令：文本查看篇

如何在 Linux 中找到并删除重复文件