<http://man.linuxde.net/objdump>

**objdump命令**是用查看目标文件或者可执行的目标文件的构成的[gcc](http://man.linuxde.net/gcc)工具。

### 选项

--archive-headers

-a

显示档案库的成员信息,类似[ls](http://man.linuxde.net/ls" \o "ls命令" \t "_blank) -l将lib\*.a的信息列出。

-b bfdname

--target=bfdname

指定目标码格式。这不是必须的，objdump能自动识别许多格式，比如：

objdump -b oasys -m vax -h fu.o

显示fu.o的头部摘要信息，明确指出该文件是Vax系统下用Oasys编译器生成的目标文件。objdump -i将给出这里可以指定的目标码格式列表。

-C

--demangle

将底层的符号名解码成用户级名字，除了去掉所开头的下划线之外，还使得C++函数名以可理解的方式显示出来。

--debugging

-g

显示调试信息。企图解析保存在文件中的调试信息并以C语言的语法显示出来。仅仅支持某些类型的调试信息。有些其他的格式被[readelf](http://man.linuxde.net/readelf" \o "readelf命令" \t "_blank) -[w](http://man.linuxde.net/w)支持。

-e

--debugging-tags

类似-g选项，但是生成的信息是和ctags工具相兼容的格式。

--disassemble

-d

从objfile中反汇编那些特定指令机器码的section。

-D

--disassemble-all

与 -d 类似，但反汇编所有section.

--prefix-addresses

反汇编的时候，显示每一行的完整地址。这是一种比较老的反汇编格式。

-EB

-EL

--endian={big|little}

指定目标文件的小端。这个项将影响反汇编出来的指令。在反汇编的文件没描述小端信息的时候用。例如S-records.

-f

--[file](http://man.linuxde.net/file)-headers

显示objfile中每个文件的整体头部摘要信息。

-h

--section-headers

--headers

显示目标文件各个section的头部摘要信息。

-H

--[help](http://man.linuxde.net/help)

简短的帮助信息。

-i

--[info](http://man.linuxde.net/info)

显示对于 -b 或者 -m 选项可用的架构和目标格式列表。

-j name

--section=name

仅仅显示指定名称为name的section的信息

-l

--line-numbers

用文件名和行号标注相应的目标代码，仅仅和-d、-D或者-r一起使用使用-[ld](http://man.linuxde.net/ld" \o "ld命令" \t "_blank)和使用-d的区别不是很大，在源码级调试的时候有用，要求编译时使用了-g之类的调试编译选项。

-m machine

--architecture=machine

指定反汇编目标文件时使用的架构，当待反汇编文件本身没描述架构信息的时候(比如S-records)，这个选项很有用。可以用-i选项列出这里能够指定的架构.

--reloc

-r

显示文件的重定位入口。如果和-d或者-D一起使用，重定位部分以反汇编后的格式显示出来。

--dynamic-reloc

-R

显示文件的动态重定位入口，仅仅对于动态目标文件意义，比如某些共享库。

-s

--full-contents

显示指定section的完整内容。默认所有的非空section都会被显示。

-S

--source

尽可能反汇编出源代码，尤其当编译的时候指定了-g这种调试参数时，效果比较明显。隐含了-d参数。

--show-raw-insn

反汇编的时候，显示每条汇编指令对应的机器码，如不指定--prefix-addresses，这将是缺省选项。

--no-show-raw-insn

反汇编时，不显示汇编指令的机器码，如不指定--prefix-addresses，这将是缺省选项。

--start-address=address

从指定地址开始显示数据，该选项影响-d、-r和-s选项的输出。

--stop-address=address

显示数据直到指定地址为止，该项影响-d、-r和-s选项的输出。

-t

--syms

显示文件的符号表入口。类似于[nm](http://man.linuxde.net/nm" \o "nm命令" \t "_blank) -s提供的信息

-T

--dynamic-syms

显示文件的动态符号表入口，仅仅对动态目标文件意义，比如某些共享库。它显示的信息类似于 nm -D|--dynamic 显示的信息。

-V

--version

版本信息

--all-headers

-x

显示所可用的头信息，包括符号表、重定位入口。-x 等价于-a -f -h -r -t 同时指定。

-z

--disassemble-zeroes

一般反汇编输出将省略大块的零，该选项使得这些零块也被反汇编。

@file 可以将选项集中到一个文件中，然后使用这个@file选项载入。

### 实例

首先，在给出后面大部分测试所基于的源代码以及编译指令。 源代码如下：

[root@localhost [test](http://man.linuxde.net/test)]# [nl](http://man.linuxde.net/nl) mytest.cpp

void printTest() {

char a;

a = 'a';

}

void printTest2() {

int a = 2;

a+=2;

}

对以上源代码进行编译，如下：

[root@localhost test]# g++ -c -g mytest.cpp

这里，生成的文件是mytest.o，为了方便测试包含了调试的信息，对可执行文件的测试，显示的结果类似。

**查看当前使用的objdump的版本号：**

[root@localhost test]# objdump -V

GNU objdump 2.17.50.0.6-14.el5 20061020

Copyright 2005 [free](http://man.linuxde.net/free) Software Foundation, Inc.

This program is free software; you may redistribute it under the terms of

the GNU General Public License. This program has absolutely no warranty.

**查看档案库文件中的信息：**

[root@localhost test]# objdump -a libmy2.a

In archive libmy2.a:

myfile.o: file format elf32-i386

rwxrwxrwx 0/0 2724 Nov 16 16:06 2009 myfile.o

mytest.o: file format elf32-i386

rw-r--r-- 0/0 727 Jul 13 15:32 2011 mytest.o

**这里，libmy2.a是一个使用ar命令将多个\*.o目标文件打包而生成的静态库。命令的输出类似ar -tv，相比较ar -tv输出如下：**

[root@localhost test]# ar -tv libmy2.a

rwxrwxrwx 0/0 2724 Nov 16 16:06 2009 myfile.o

rw-r--r-- 0/0 727 Jul 13 15:32 2011 mytest.o

显示可用的架构和目标结构列表：

[root@localhost test]# objdump -i

BFD header file version 2.17.50.0.6-14.el5 20061020

elf32-i386

(header little endian, data little endian)

i386

a.out-i386-linux

(header little endian, data little endian)

i386

efi-app-ia32

(header little endian, data little endian)

i386

elf64-x86-64

(header little endian, data little endian)

i386

elf64-little

(header little endian, data little endian)

i386

elf64-big

(header big endian, data big endian)

i386

elf32-little

(header little endian, data little endian)

i386

elf32-big

(header big endian, data big endian)

i386

srec

(header endianness unknown, data endianness unknown)

i386

symbolsrec

(header endianness unknown, data endianness unknown)

i386

tekhex

(header endianness unknown, data endianness unknown)

i386

binary

(header endianness unknown, data endianness unknown)

i386

ihex

(header endianness unknown, data endianness unknown)

i386

trad-core

(header endianness unknown, data endianness unknown)

elf32-i386 a.out-i386-linux efi-app-ia32 elf64-x86-64

i386 elf32-i386 a.out-i386-linux efi-app-ia32 elf64-x86-64

elf64-little elf64-big elf32-little elf32-big srec symbolsrec

i386 elf64-little elf64-big elf32-little elf32-big srec symbolsrec

tekhex binary ihex trad-core

i386 tekhex binary ihex ---------

这里，显示的信息是相对于 -b 或者 -m 选项可用的架构和目标格式列表。

**显示mytest.o文件中的text段的内容：**

[root@localhost test]# objdump --section=.text -s mytest.o

mytest.o: file format elf32-i386

Contents of section .text:

0000 5589e583 ec10c645 ff61c9c3 5589e583 U......E.a..U...

0010 ec10c745 fc020000 008345fc 02c9c3 ...E......E....

这里注意，不能单独使用-j或者--section，例如objdump --section=.text mytest.o是不会运行成功的。

**反汇编mytest.o中的text段内容，并尽可能用源代码形式表示：**

[root@localhost test]# objdump -j .text -S mytest.o

mytest.o: file format elf32-i386

Disassembly of section .text:

00000000 <\_Z9printTestv>:

void printTest()

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 83 ec 10 sub $0x10,%esp

{

char a;

a = 'a';

6: c6 45 ff 61 movb $0x61,0xffffffff(%ebp)

}

a: c9 leave

b: c3 ret

000000c <\_Z10printTest2v>:

void printTest2()

c: 55 push %ebp

d: 89 e5 mov %esp,%ebp

f: 83 ec 10 sub $0x10,%esp

{

int a = 2;

12: c7 45 [fc](http://man.linuxde.net/fc) 02 00 00 00 movl $0x2,0xfffffffc(%ebp)

a+=2;

19: 83 45 fc 02 addl $0x2,0xfffffffc(%ebp)

}

1d: c9 leave

1e: c3 ret

这里注意，不能单独使用-j或者--section，例如objdump -j .text mytest.o是不会运行成功的。另外-S命令对于包含调试信息的目标文件，显示的效果比较好，如果编译时没有指定g++的-g选项，那么目标文件就不包含调试信息，那么显示效果就差多了。

**反汇编出mytest.o的源代码:**

[root@localhost test]# objdump -S mytest.o

mytest.o: file format elf32-i386

Disassembly of section .text:

00000000 <\_Z9printTestv>:

void printTest()

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 83 ec 10 sub $0x10,%esp

{

char a;

a = 'a';

6: c6 45 ff 61 movb $0x61,0xffffffff(%ebp)

}

a: c9 leave

b: c3 ret

0000000c <\_Z10printTest2v>:

void printTest2()

c: 55 push %ebp

d: 89 e5 mov %esp,%ebp

f: 83 ec 10 sub $0x10,%esp

{

int a = 2;

12: c7 45 fc 02 00 00 00 movl $0x2,0xfffffffc(%ebp)

a+=2;

19: 83 45 fc 02 addl $0x2,0xfffffffc(%ebp)

}

1d: c9 leave

1e: c3 ret

这里，尤其当编译的时候指定了-g这种调试参数时，反汇编的效果比较明显。隐含了-d参数。

**显示文件的符号表入口:**

[root@localhost test]# objdump -t mytest.o

mytest.o: file format elf32-i386

SYMBOL TABLE:

00000000 l [df](http://man.linuxde.net/df) \*ABS\* 00000000 mytest.cpp

00000000 l d .text 00000000 .text

00000000 l d .data 00000000 .data

00000000 l d .bss 00000000 .bss

00000000 l d .debug\_abbrev 00000000 .debug\_abbrev

00000000 l d .debug\_info 00000000 .debug\_info

00000000 l d .debug\_line 00000000 .debug\_line

00000000 l d .debug\_frame 00000000 .debug\_frame

00000000 l d .debug\_loc 00000000 .debug\_loc

00000000 l d .debug\_pubnames 00000000 .debug\_pubnames

00000000 l d .debug\_aranges 00000000 .debug\_aranges

00000000 l d .note.GNU-stack 00000000 .note.GNU-stack

00000000 l d .comment 00000000 .comment

00000000 g F .text 0000000c \_Z9printTestv

00000000 \*UND\* 00000000 \_\_gxx\_personality\_v0

0000000c g F .text 00000013 \_Z10printTest2v

这里，输出的信息类似nm -s命令的输出，相比较之下，nm命令的输出如下：

[root@localhost test]# nm -s mytest.o

0000000c T \_Z10printTest2v

00000000 T \_Z9printTestv

U \_\_gxx\_personality\_v0

**显示文件的符号表入口，将底层符号解码并表示成用户级别:**

[root@localhost test]# objdump -t -C mytest.o

mytest.o: file format elf32-i386

SYMBOL TABLE:

00000000 l df \*ABS\* 00000000 mytest.cpp

00000000 l d .text 00000000 .text

00000000 l d .data 00000000 .data

00000000 l d .bss 00000000 .bss

00000000 l d .debug\_abbrev 00000000 .debug\_abbrev

00000000 l d .debug\_info 00000000 .debug\_info

00000000 l d .debug\_line 00000000 .debug\_line

00000000 l d .debug\_frame 00000000 .debug\_frame

00000000 l d .debug\_loc 00000000 .debug\_loc

00000000 l d .debug\_pubnames 00000000 .debug\_pubnames

00000000 l d .debug\_aranges 00000000 .debug\_aranges

00000000 l d .note.GNU-stack 00000000 .note.GNU-stack

00000000 l d .comment 00000000 .comment

00000000 g F .text 0000000c printTest()

00000000 \*UND\* 00000000 \_\_gxx\_personality\_v0

0000000c g F .text 00000013 printTest2()

这里，和没-C相比，printTest2函数可读性增加了。

**反汇编目标文件的特定机器码段：**

[root@localhost test]# objdump -d mytest.o

mytest.o: file format elf32-i386

Disassembly of section .text:

00000000 <\_Z9printTestv>:

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 83 ec 10 sub $0x10,%esp

6: c6 45 ff 61 movb $0x61,0xffffffff(%ebp)

a: c9 leave

b: c3 ret

0000000c <\_Z10printTest2v>:

c: 55 push %ebp

d: 89 e5 mov %esp,%ebp

f: 83 ec 10 sub $0x10,%esp

12: c7 45 fc 02 00 00 00 movl $0x2,0xfffffffc(%ebp)

19: 83 45 fc 02 addl $0x2,0xfffffffc(%ebp)

1d: c9 leave

1e: c3 ret

这里，对text段的内容进行了反汇编。

**反汇编特定段，并将汇编代码对应的文件名称和行号对应上：**

[root@localhost test]# objdump -d -l mytest.o

mytest.o: file format elf32-i386

Disassembly of section .text:

00000000 <\_Z9printTestv>:

\_Z9printTestv():

/root/test/04\_libraryTest/mytest.cpp:1

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 83 ec 10 sub $0x10,%esp

/root/test/04\_libraryTest/mytest.cpp:4

6: c6 45 ff 61 movb $0x61,0xffffffff(%ebp)

/root/test/04\_libraryTest/mytest.cpp:5

a: c9 leave

b: c3 ret

0000000c <\_Z10printTest2v>:

\_Z10printTest2v():

/root/test/04\_libraryTest/mytest.cpp:6

c: 55 push %ebp

d: 89 e5 mov %esp,%ebp

f: 83 ec 10 sub $0x10,%esp

/root/test/04\_libraryTest/mytest.cpp:8

12: c7 45 fc 02 00 00 00 movl $0x2,0xfffffffc(%ebp)

/root/test/04\_libraryTest/mytest.cpp:9

19: 83 45 fc 02 addl $0x2,0xfffffffc(%ebp)

/root/test/04\_libraryTest/mytest.cpp:10

1d: c9 leave

1e: c3 ret

这里，项"-d"从objfile中反汇编那些特定指令机器码的section，而使用"-l"指定用文件名和行号标注相应的目标代码，仅仅和-d、-D或者-r一起使用，使用-ld和使用-d的区别不是很大，在源码级调试的时候有用，要求编译时使用了-g之类的调试编译选项。

**显示目标文件各个段的头部摘要信息：**

[root@localhost test]# objdump -h mytest.o

mytest.o: file format elf32-i386

Sections:

Idx Name Size VMA LMA File off Algn

0 .text 0000001f 00000000 00000000 00000034 2\*\*2

CONTENTS, ALLOC, LOAD, [readonly](http://man.linuxde.net/readonly), CODE

1 .data 00000000 00000000 00000000 00000054 2\*\*2

CONTENTS, ALLOC, LOAD, DATA

2 .bss 00000000 00000000 00000000 00000054 2\*\*2

ALLOC

3 .debug\_abbrev 00000046 00000000 00000000 00000054 2\*\*0

CONTENTS, READONLY, DEBUGGING

4 .debug\_info 000000ed 00000000 00000000 0000009a 2\*\*0

CONTENTS, RELOC, READONLY, DEBUGGING

5 .debug\_line 0000003e 00000000 00000000 00000187 2\*\*0

CONTENTS, RELOC, READONLY, DEBUGGING

6 .debug\_frame 00000044 00000000 00000000 000001c8 2\*\*2

CONTENTS, RELOC, READONLY, DEBUGGING

7 .debug\_loc 00000058 00000000 00000000 0000020c 2\*\*0

CONTENTS, READONLY, DEBUGGING

8 .debug\_pubnames 0000002f 00000000 00000000 00000264 2\*\*0

CONTENTS, RELOC, READONLY, DEBUGGING

9 .debug\_aranges 00000020 00000000 00000000 00000293 2\*\*0

CONTENTS, RELOC, READONLY, DEBUGGING

10 .comment 0000002e 00000000 00000000 000002b3 2\*\*0

CONTENTS, READONLY

11 .note.GNU-stack 00000000 00000000 00000000 000002e1 2\*\*0

CONTENTS, READONLY

这里，更多的内容参见[man](http://man.linuxde.net/man) objdump中的这个选项。