Linux Objdump Command Examples (Disassemble a Binary File)

by Himanshu Arora on September 21, 2012

Objdump command in Linux is used to provide thorough information on object files. This command is mainly used by the programmers who work on compilers, but still its a very handy tool for normal programmers also when it comes to debugging. In this article, we will understand how to use objdump command through some examples.

Basic syntax of objdump is :

```
objdump [options] objfile...
```

There is a wide range of options available for this command. We will try to cover a good amount of them in this tutorial.

Examples

The ELF binary file of the following C program is used in all the examples mentioned in this article.

```
#include<stdio.h>

int main(void)
{
    int n = 6;
    float f=1;
    int i = 1;
    for(;i<=n;i++)
        f=f*i;
    printf("\n Factorial is : [%f]\n",f);
    return 0;
}
```

Note: The above is just a test code that was being used for some other purpose, but I found it simple enough to use for this article.

1. Display the contents of the overall file header using -f option

Consider the following example :

```
$ objdump -f factorial

factorial:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400440
```

So we see that the information related to the overall file header was shown in the output.

NOTE: The executable format used in the examples is ELF. To know more about it, refer to our article on ELF file format.

2.Display object format specific file header contents using -p option

The following example prints the object file format specific information.

```
$ objdump -p factorial

factorial:     file format elf64-x86-64

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000400040 paddr 0x0000000000400040 align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off    0x0000000000000238 vaddr 0x0000000000400238 paddr 0x0000000000400238 align 2**0
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000400000 paddr 0x0000000000400000 align 2**21
         filesz 0x0000000000000734 memsz 0x0000000000000734 flags r-x
    LOAD off    0x0000000000000e18 vaddr 0x0000000000600e18 paddr 0x0000000000600e18 align 2**21
         filesz 0x0000000000000208 memsz 0x0000000000000218 flags rw-
 DYNAMIC off    0x0000000000000e40 vaddr 0x0000000000600e40 paddr 0x0000000000600e40 align 2**3
         filesz 0x00000000000001a0 memsz 0x00000000000001a0 flags rw-
 ..
Dynamic Section:
  NEEDED               libc.so.6
  INIT                 0x00000000004003f0
  FINI                 0x0000000000400668
  HASH                 0x0000000000400298
```

```
  GNU_HASH             0x00000000004002c0
  STRTAB               0x0000000000400340
  SYMTAB               0x00000000004002e0
  STRSZ                0x000000000000003f
  SYMENT               0x0000000000000018
  DEBUG                0x0000000000000000
  PLTGOT               0x0000000000600fe8
  ..
Version References:
  required from libc.so.6:
    0x09691a75 0x00 02 GLIBC_2.2.5
```

3. Display the contents of the section headers using -h option

There can be various sections in an object file. Information related to them can be printed using -h option.

The following examples shows various sections. As you see there are total of 26 (only partial output is shown here).

```
$ objdump -h factorial

factorial:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000400238  0000000000400238  00000238  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000020  0000000000400254  0000000000400254  00000254  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000400274  0000000000400274  00000274  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .hash         00000024  0000000000400298  0000000000400298  00000298  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  ....
 14 .fini         0000000e  0000000000400668  0000000000400668  00000668  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .rodata       0000001b  0000000000400678  0000000000400678  00000678  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 16 .eh_frame_hdr 00000024  0000000000400694  0000000000400694  00000694  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 17 .eh_frame     0000007c  00000000004006b8  00000000004006b8  000006b8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 18 .ctors        00000010  0000000000600e18  0000000000600e18  00000e18  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 19 .dtors        00000010  0000000000600e28  0000000000600e28  00000e28  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 ...
 23 .got.plt      00000028  0000000000600fe8  0000000000600fe8  00000fe8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .data         00000010  0000000000601010  0000000000601010  00001010  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 25 .bss          00000010  0000000000601020  0000000000601020  00001020  2**3
                  ALLOC
 26 .comment      00000023  0000000000000000  0000000000000000  00001020  2**0
                  CONTENTS, READONLY
```

So we see that the information related to all the section headers was printed in the output. In the output above, Size is the size of the loaded section, VMA represents the virtual memory address, LMA represents the logical memory address, File off is this section's offset from the beginning of the file, Algn represents alignment, CONTENTS, ALLOC, LOAD, READONLY, DATA are flags that represent that a particular section is to be LOADED or is READONLY etc.

4. Display the contents of all headers using -x option

Information related to all the headers in the object file can be retrieved using the -x option.

The following example displays all the sections (only partial output is shown here):

```
$ objdump -x factorial

factorial:     file format elf64-x86-64
factorial
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400440

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000400040 paddr 0x0000000000400040 align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off    0x0000000000000238 vaddr 0x0000000000400238 paddr 0x0000000000400238 align 2**0
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
  .....
EH_FRAME off    0x0000000000000694 vaddr 0x0000000000400694 paddr 0x0000000000400694 align 2**2
         filesz 0x0000000000000024 memsz 0x0000000000000024 flags r--
   STACK off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**3
         filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-
   RELRO off    0x0000000000000e18 vaddr 0x0000000000600e18 paddr 0x0000000000600e18 align 2**0
         filesz 0x00000000000001e8 memsz 0x00000000000001e8 flags r--

Dynamic Section:
  NEEDED               libc.so.6
  INIT                 0x00000000004003f0
  FINI                 0x0000000000400668
  HASH                 0x0000000000400298
  GNU_HASH             0x00000000004002c0
  STRTAB               0x0000000000400340
  SYMTAB               0x00000000004002e0
  STRSZ                0x000000000000003f
....
Version References:
  required from libc.so.6:
    0x09691a75 0x00 02 GLIBC_2.2.5

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000400238  0000000000400238  00000238  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000020  0000000000400254  0000000000400254  00000254  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000400274  0000000000400274  00000274  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .hash         00000024  0000000000400298  0000000000400298  00000298  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .gnu.hash     0000001c  00000000004002c0  00000000004002c0  000002c0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  .....
 18 .ctors        00000010  0000000000600e18  0000000000600e18  00000e18  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 19 .dtors        00000010  0000000000600e28  0000000000600e28  00000e28  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 20 .jcr          00000008  0000000000600e38  0000000000600e38  00000e38  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 21 .dynamic      000001a0  0000000000600e40  0000000000600e40  00000e40  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 22 .got          00000008  0000000000600fe0  0000000000600fe0  00000fe0  2**3
```

```
                  CONTENTS, ALLOC, LOAD, DATA
 23 .got.plt      00000028  0000000000600fe8  0000000000600fe8  00000fe8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .data         00000010  0000000000601010  0000000000601010  00001010  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 25 .bss          00000010  0000000000601020  0000000000601020  00001020  2**3
                  ALLOC
 26 .comment      00000023  0000000000000000  0000000000000000  00001020  2**0
                  CONTENTS, READONLY
SYMBOL TABLE:
0000000000400238 l    d  .interp        0000000000000000              .interp
0000000000400254 l    d  .note.ABI-tag  0000000000000000              .note.ABI-tag
0000000000400274 l    d  .note.gnu.build-id     0000000000000000              .note.gnu.build-id
0000000000400298 l    d  .hash  0000000000000000              .hash
00000000004002c0 l    d  .gnu.hash      0000000000000000              .gnu.hash
00000000004002e0 l    d  .dynsym        0000000000000000              .dynsym
0000000000400340 l    d  .dynstr        0000000000000000              .dynstr
0000000000400380 l    d  .gnu.version   0000000000000000              .gnu.version
0000000000400388 l    d  .gnu.version_r 0000000000000000              .gnu.version_r
....
0000000000600e30 g     O .dtors 0000000000000000              .hidden __DTOR_END__
00000000004005a0 g     F .text  0000000000000089              __libc_csu_init
0000000000601020 g       *ABS*  0000000000000000              __bss_start
0000000000601030 g       *ABS*  0000000000000000              _end
0000000000601020 g       *ABS*  0000000000000000              _edata
0000000000400524 g     F .text  0000000000000060              main
00000000004003f0 g     F .init  0000000000000000              _init
```

5. Display assembler contents of executable sections using -d option

Consider the following example. The assembler contents of executable sections (in the object file) are displayed in this output (partial output shown below):

```
$ objdump -d factorial

factorial:     file format elf64-x86-64

Disassembly of section .init:

00000000004003f0 :
  4003f0:       48 83 ec 08             sub    $0x8,%rsp
  4003f4:       e8 73 00 00 00          callq  40046c
  ..
Disassembly of section .plt:

0000000000400408 :
  400408:       ff 35 e2 0b 20 00       pushq  0x200be2(%rip)        # 600ff0
  40040e:       ff 25 e4 0b 20 00       jmpq   *0x200be4(%rip)        # 600ff8
  400414:       0f 1f 40 00             nopl   0x0(%rax)

0000000000400418 :
  400418:       ff 25 e2 0b 20 00       jmpq   *0x200be2(%rip)        # 601000
  40041e:       68 00 00 00 00          pushq  $0x0
  400423:       e9 e0 ff ff ff          jmpq   400408

0000000000400428 :
  400428:       ff 25 da 0b 20 00       jmpq   *0x200bda(%rip)        # 601008
  40042e:       68 01 00 00 00          pushq  $0x1
  400433:       e9 d0 ff ff ff          jmpq   400408

Disassembly of section .text:

0000000000400440 :
```

```
  400440:       31 ed                   xor    %ebp,%ebp
  400442:       49 89 d1                mov    %rdx,%r9
  400445:       5e                      pop    %rsi
 ...
000000000040046c :
  40046c:       48 83 ec 08             sub    $0x8,%rsp
  400470:       48 8b 05 69 0b 20 00    mov    0x200b69(%rip),%rax        # 600fe0
  400477:       48 85 c0                test   %rax,%rax
  40047a:       74 02                   je     40047e
  40047c:       ff d0                   callq  *%rax
  ...
0000000000400490 :
  400490:       55                      push   %rbp
  400491:       48 89 e5                mov    %rsp,%rbp
  400494:       53                      push   %rbx
  400495:       48 83 ec 08             sub    $0x8,%rsp
  400499:       80 3d 80 0b 20 00 00    cmpb   $0x0,0x200b80(%rip)        # 601020
  4004a0:       75 4b                   jne    4004ed
  4004a2:       bb 30 0e 60 00          mov    $0x600e30,%ebx
  4004fb:       00 00 00 00 00
  ...
0000000000400500 :
  400500:       55                      push   %rbp
  400501:       48 83 3d 2f 09 20 00    cmpq   $0x0,0x20092f(%rip)        # 600e38
  400508:       00
  400509:       48 89 e5                mov    %rsp,%rbp
  40050c:       74 12                   je     400520
  40050e:       b8 00 00 00 00          mov    $0x0,%eax
  400513:       48 85 c0                test   %rax,%rax
  400516:       74 08                   je     400520
  400518:       bf 38 0e 60 00          mov    $0x600e38,%edi
  40051d:       c9                      leaveq
  40051e:       ff e0                   jmpq   *%rax
  400520:       c9                      leaveq
  400521:       c3                      retq
  400522:       90                      nop
  400523:       90                      nop

0000000000400524 :
  400524:       55                      push   %rbp
  400525:       48 89 e5                mov    %rsp,%rbp
  400528:       48 83 ec 10             sub    $0x10,%rsp
  40052c:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%rbp)
  400533:       b8 00 00 80 3f          mov    $0x3f800000,%eax
  400538:       89 45 f8                mov    %eax,-0x8(%rbp)
 ...

Disassembly of section .fini:

0000000000400668 :
  400668:       48 83 ec 08             sub    $0x8,%rsp
  40066c:       e8 1f fe ff ff          callq  400490
  400671:       48 83 c4 08             add    $0x8,%rsp
  400675:       c3                      retq
```

6. Display assembler contents of all sections using -D option

In case the assembler contents of all the sections is required in output, the option -D can be used.

Consider the following output :

```
$ objdump -D factorial | pager
```

```
factorial:     file format elf64-x86-64

Disassembly of section .interp:

0000000000400238 :
  400238:       2f                      (bad)
  400239:       6c                      insb   (%dx),%es:(%rdi)
  40023a:       69 62 36 34 2f 6c 64    imul   $0x646c2f34,0x36(%rdx),%esp
  400241:       2d 6c 69 6e 75          sub    $0x756e696c,%eax
  400246:       78 2d                   js     400275
  400248:       78 38                   js     400282
  40024a:       36                      ss
  40024b:       2d 36 34 2e 73          sub    $0x732e3436,%eax
  400250:       6f                      outsl  %ds:(%rsi),(%dx)
  400251:       2e 32 00                xor    %cs:(%rax),%al

Disassembly of section .note.ABI-tag:

0000000000400254 :
  400254:       04 00                   add    $0x0,%al
  400256:       00 00                   add    %al,(%rax)
  400258:       10 00                   adc    %al,(%rax)
  40025a:       00 00                   add    %al,(%rax)
  40025c:       01 00                   add    %eax,(%rax)
  40025e:       00 00                   add    %al,(%rax)
  400260:       47                      rex.RXB
  400261:       4e 55                   rex.WRX push   %rbp
  400263:       00 00                   add    %al,(%rax)
  400265:       00 00                   add    %al,(%rax)
  400267:       00 02                   add    %al,(%rdx)
  400269:       00 00                   add    %al,(%rax)
  40026b:       00 06                   add    %al,(%rsi)
  40026d:       00 00                   add    %al,(%rax)
  40026f:       00 0f                   add    %cl,(%rdi)
  400271:       00 00                   add    %al,(%rax)
        ...
        ...
        ...
```

So we see that the relevant output was displayed. Since the output was very long, so I clipped it. Note that I used the pager command for controlling the output.

7. Display the full contents of all sections using -s option

Consider the following example :

```
$ objdump -s factorial

factorial:     file format elf64-x86-64

Contents of section .interp:
 400238 2f6c6962 36342f6c 642d6c69 6e75782d  /lib64/ld-linux-
 400248 7838362d 36342e73 6f2e3200           x86-64.so.2.
Contents of section .note.ABI-tag:
 400254 04000000 10000000 01000000 474e5500  ............GNU.
 400264 00000000 02000000 06000000 0f000000  ................
Contents of section .note.gnu.build-id:
 400274 04000000 14000000 03000000 474e5500  ............GNU.
 400284 c6928568 6751d6de 6ddd2eb1 7c5cd0ff  ...hgQ..m...|\..
 400294 670751c6                             g.Q.
Contents of section .hash:
 400298 03000000 04000000 02000000 03000000  ................
```

```
 4002a8 01000000 00000000 00000000 00000000  ................
 4002b8 00000000                             ....
Contents of section .gnu.hash:
 4002c0 01000000 01000000 01000000 00000000  ................
 4002d0 00000000 00000000 00000000           ............
Contents of section .dynsym:
 4002e0 00000000 00000000 00000000 00000000  ................
 4002f0 00000000 00000000 1a000000 12000000  ................
 400300 00000000 00000000 00000000 00000000  ................
 400310 01000000 20000000 00000000 00000000  .... ...........
 400320 00000000 00000000 21000000 12000000  ........!.......
 400330 00000000 00000000 00000000 00000000  ................
Contents of section .dynstr:
 400340 005f5f67 6d6f6e5f 73746172 745f5f00  .__gmon_start__.
 400350 6c696263 2e736f2e 36007072 696e7466  libc.so.6.printf
 400360 005f5f6c 6962635f 73746172 745f6d61  .__libc_start_ma
 400370 696e0047 4c494243 5f322e32 2e3500    in.GLIBC_2.2.5.
Contents of section .gnu.version:
 400380 00000200 00000200                    ........
Contents of section .gnu.version_r:
 400388 01000100 10000000 10000000 00000000  ................
 400398 751a6909 00000200 33000000 00000000  u.i.....3.......
Contents of section .rela.dyn:
 4003a8 e00f6000 00000000 06000000 02000000  ..`.............
 4003b8 00000000 00000000                    ........
Contents of section .rela.plt:
 4003c0 00106000 00000000 07000000 01000000  ..`.............
 4003d0 00000000 00000000 08106000 00000000  ..........`.....
 4003e0 07000000 03000000 00000000 00000000  ................
Contents of section .init:
 4003f0 4883ec08 e8730000 00e80201 0000e82d  H....s.........-
 400400 02000048 83c408c3                    ...H....
Contents of section .plt:
 400408 ff35e20b 2000ff25 e40b2000 0f1f4000  .5.. ..%.. ...@.
 400418 ff25e20b 20006800 000000e9 e0ffffff  .%.. .h........
 400428 ff25da0b 20006801 000000e9 d0ffffff  .%.. .h........
Contents of section .text:
 400440 31ed4989 d15e4889 e24883e4 f0505449  1.I..^H..H...PTI
 400450 c7c09005 400048c7 c1a00540 0048c7c7  ....@.H....@.H..
 400460 24054000 e8bfffff fff49090 4883ec08  $.@.........H...
 400470 488b0569 0b200048 85c07402 ffd04883  H..i. .H..t...H.
 400480 c408c390 90909090 90909090 90909090  ................
 400490 554889e5 534883ec 08803d80 0b200000  UH..SH....=.. ..
 ....
 4005e0 e80bfeff ff4885ed 741c31db 0f1f4000  .....H..t.1...@.
 4005f0 4c89fa4c 89f64489 ef41ff14 dc4883c3  L..L..D..A...H..
 400600 014839eb 72ea488b 5c240848 8b6c2410  .H9.r.H.\$.H.l$.
 400610 4c8b6424 184c8b6c 24204c8b 7424284c  L.d$.L.l$ L.t$(L
 400620 8b7c2430 4883c438 c3909090 90909090  .|$0H..8........
 400630 554889e5 534883ec 08488b05 d8072000  UH..SH...H.... .
 400640 4883f8ff 7419bb18 0e60000f 1f440000  H...t....`...D..
 400650 4883eb08 ffd0488b 034883f8 ff75f148  H.....H..H...u.H
 400660 83c4085b c9c39090                    ...[....
Contents of section .fini:
 400668 4883ec08 e81ffeff ff4883c4 08c3      H........H....
Contents of section .rodata:
 400678 01000200 0a204661 63746f72 69616c20  ..... Factorial
 400688 6973203a 205b2566 5d0a00             is : [%f]..
Contents of section .eh_frame_hdr:
 400694 011b033b 20000000 03000000 90feffff  ...; ...........
 4006a4 3c000000 fcfeffff 5c000000 0cffffff
```

So we see that the complete contents for all the sections were displayed in the output.

8. Display debug information using -g option

Consider the following example:

```
$ objdump -g factorial

factorial:     file format elf64-x86-64
```

So we see that all the available debug information was printed in output.

9. Display the contents of symbol table (or tables) using the -t option

Consider the following example :

```
$ objdump -t factorial

factorial:     file format elf64-x86-64

SYMBOL TABLE:
0000000000400238 l    d  .interp        0000000000000000              .interp
0000000000400254 l    d  .note.ABI-tag  0000000000000000              .note.ABI-tag
0000000000400274 l    d  .note.gnu.build-id     0000000000000000              .note.gnu.build-id
0000000000400298 l    d  .hash  0000000000000000              .hash
00000000004002c0 l    d  .gnu.hash      0000000000000000              .gnu.hash
00000000004002e0 l    d  .dynsym        0000000000000000              .dynsym
0000000000400340 l    d  .dynstr        0000000000000000              .dynstr
0000000000400380 l    d  .gnu.version   0000000000000000              .gnu.version
.....
0000000000601010 g       .data  0000000000000000              __data_start
0000000000601018 g     O .data  0000000000000000              .hidden __dso_handle
0000000000600e30 g     O .dtors 0000000000000000              .hidden __DTOR_END__
00000000004005a0 g     F .text  0000000000000089              __libc_csu_init
0000000000601020 g       *ABS*  0000000000000000              __bss_start
0000000000601030 g       *ABS*  0000000000000000              _end
0000000000601020 g       *ABS*  0000000000000000              _edata
0000000000400524 g     F .text  0000000000000060              main
00000000004003f0 g     F .init  0000000000000000              _init
```

So we see that the contents of symbol table were displayed in the output.

10. Display the contents of dynamic symbol table using -T option

Dynamic symbols are those which are resolved during run time. The information related to these symbols can be retrieved using the -D option.

Consider the following example :

```
$ objdump -T factorial

factorial:     file format elf64-x86-64

DYNAMIC SYMBOL TABLE:
0000000000000000      DF *UND*  0000000000000000  GLIBC_2.2.5 printf
0000000000000000  w   D  *UND*  0000000000000000              __gmon_start__
0000000000000000      DF *UND*  0000000000000000  GLIBC_2.2.5 __libc_start_main
```

So we see that information related to dynamic symbols was displayed in output.

11. Display the dynamic relocation entries in the file using -R option

Consider the following example:

```
$ objdump -R factorial

factorial:     file format elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
0000000000600fe0 R_X86_64_GLOB_DAT  __gmon_start__
0000000000601000 R_X86_64_JUMP_SLOT  printf
0000000000601008 R_X86_64_JUMP_SLOT  __libc_start_main
```

So we see that all the dynamic relocation entries were displayed in the output.

12. Display section of interest using -j option

This is extremely useful when you know the section related to which the information is required. The option -j is used in this case.

Consider the following example :

```
$ objdump -s -j.rodata factorial

factorial:     file format elf64-x86-64

Contents of section .rodata:
 400678 01000200 0a204661 63746f72 69616c20  ..... Factorial
 400688 6973203a 205b2566 5d0a00             is : [%f]..
```

So we see that information related to rodata section was displayed above.

13. Use the older disassembly format using –prefix-addresses option

The older format prints the complete address on each line.

Consider the following example :

```
$ objdump  -D --prefix-addresses factorial

factorial:     file format elf64-x86-64
Disassembly of section .interp:
0000000000400238 <.interp> (bad)
0000000000400239 <.interp+0x1> insb   (%dx),%es:(%rdi)
000000000040023a <.interp+0x2> imul   $0x646c2f34,0x36(%rdx),%esp
0000000000400241 <.interp+0x9> sub    $0x756e696c,%eax
0000000000400246 <.interp+0xe> js     0000000000400275 <_init-0x17b>
0000000000400248 <.interp+0x10> js     0000000000400282 <_init-0x16e>
000000000040024a <.interp+0x12> ss
000000000040024b <.interp+0x13> sub    $0x732e3436,%eax
0000000000400250 <.interp+0x18> outsl  %ds:(%rsi),(%dx)
0000000000400251 <.interp+0x19> xor    %cs:(%rax),%al

Disassembly of section .note.ABI-tag:
0000000000400254 <.note.ABI-tag> add    $0x0,%al
0000000000400256 <.note.ABI-tag+0x2> add    %al,(%rax)
0000000000400258 <.note.ABI-tag+0x4> adc    %al,(%rax)
000000000040025a <.note.ABI-tag+0x6> add    %al,(%rax)
000000000040025c <.note.ABI-tag+0x8> add    %eax,(%rax)
```

```
000000000040025e <.note.ABI-tag+0xa> add    %al,(%rax)
0000000000400260 <.note.ABI-tag+0xc> rex.RXB
0000000000400261 <.note.ABI-tag+0xd> rex.WRX push   %rbp
0000000000400263 <.note.ABI-tag+0xf> add    %al,(%rax)
0000000000400265 <.note.ABI-tag+0x11> add    %al,(%rax)
0000000000400267 <.note.ABI-tag+0x13> add    %al,(%rdx)
0000000000400269 <.note.ABI-tag+0x15> add    %al,(%rax)
000000000040026b <.note.ABI-tag+0x17> add    %al,(%rsi)
000000000040026d <.note.ABI-tag+0x19> add    %al,(%rax)
000000000040026f <.note.ABI-tag+0x1b> add    %cl,(%rdi)
0000000000400271 <.note.ABI-tag+0x1d> add    %al,(%rax)
        ...

Disassembly of section .note.gnu.build-id:
0000000000400274 <.note.gnu.build-id> add    $0x0,%al
0000000000400276 <.note.gnu.build-id+0x2> add    %al,(%rax)
0000000000400278 <.note.gnu.build-id+0x4> adc    $0x0,%al
...
...
...
```

So we see that complete address were printed in the output.

14. Accept input options from a file using @ option

If you want, the options to objdump can be read from a file. This can be done using '@' option.

Consider the following example :

```
$ objdump -v -i
GNU objdump (GNU Binutils for Ubuntu) 2.20.1-system.20100303
Copyright 2009 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
```

In this example above, I have used the -v and -i options. While -v is used to print the version information, -i is used to provide supported object formats and architectures.

Now I created a file and add these two options there.

```
$ cat options.txt
-v -i
```

Execute the objdump by calling the options.txt file as shown below. This displays the same output as above, as it is reading the options from the options.txt file.

```
$ objdump @options.txt
GNU objdump (GNU Binutils for Ubuntu) 2.20.1-system.20100303
Copyright 2009 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
```

> Add your comment

If you enjoyed this article, you might also like..

1. 50 Linux Sysadmin Tutorials
2. 50 Most Frequently Used Linux Commands (With Examples)
3. Top 25 Best Linux Performance Monitoring and Debugging Tools
4. Mommy, I found it! – 15 Practical Linux Find Command Examples
5. Linux 101 Hacks 2nd Edition eBook

- Awk Introduction – 7 Awk Print Examples
- Advanced Sed Substitution Examples
- 8 Essential Vim Editor Navigation Fundamentals
- 25 Most Frequently Used Linux IPTables Rules Examples
- Turbocharge PuTTY with 12 Powerful Add-Ons

Linux 101 Hacks Book

Bash 101 Hacks Book Sed and Awk 101 Hacks Book Nagios Core 3 Book Vim 101 Hacks Book

Tagged as: ELF Header, Objdump ARM, Objdump for Windows

{ 4 comments… add one }

- Athul September 22, 2012, 1:19 am

  Since now ELF is the format standard , isnt "readelf" a flexible command for program and section headers ? objdump is anyway best suited for disassemble object code .

  Link
- m4tch3t September 22, 2012, 8:52 pm

  you must clarify what connection between those adressess (logical or virtual) with the label,

  like:
  0 .interp 0000001c 0000000000400238 0000000000400238
  how to read and associate between .interp and 0000000000400238..
  i'm a newbie and i want to learn…^_^

  Link
- shadi April 23, 2015, 8:46 pm

  Is there a way to find out which part of the instructions, i.e. which addresses, from assembler contents of executable sections are from each C code and which part of it is from linking dynamic libraries?

  Link
- shadi April 23, 2015, 9:46 pm

  I meant static libraries!

  Link

Leave a Comment

Name

Email

Website

Comment

Submit

☐ Notify me of followup comments via e-mail

Next post: Linux Groff Command Examples to Create Formatted Document

Previous post: 15 SQLite3 SQL Commands Explained with Examples

RSS | Email | Twitter | Facebook | Google+

Custom Search Search

EBOOKS

- Linux 101 Hacks Book Linux 101 Hacks 2nd Edition eBook - Practical Examples to Build a Strong Foundation in Linux
- Bash 101 Hacks eBook - Take Control of Your Bash Command Line and Shell Scripting
- Sed and Awk 101 Hacks eBook - Enhance Your UNIX / Linux Life with Sed and Awk
- Vim 101 Hacks eBook - Practical Examples for Becoming Fast and Productive in Vim Editor
- Nagios Core 3 eBook - Monitor Everything, Be Proactive, and Sleep Well

POPULAR POSTS

- 15 Essential Accessories for Your Nikon or Canon DSLR Camera
- 12 Amazing and Essential Linux Books To Enrich Your Brain and Library
- 50 UNIX / Linux Sysadmin Tutorials
- 50 Most Frequently Used UNIX / Linux Commands (With Examples)
- How To Be Productive and Get Things Done Using GTD
- 30 Things To Do When you are Bored and have a Computer
- Linux Directory Structure (File System Structure) Explained with Examples
- Linux Crontab: 15 Awesome Cron Job Examples
- Get a Grip on the Grep! – 15 Practical Grep Command Examples
- Unix LS Command: 15 Practical Examples
- 15 Examples To Master Linux Command Line History
- Top 10 Open Source Bug Tracking System
- Vi and Vim Macro Tutorial: How To Record and Play
- Mommy, I found it! -- 15 Practical Linux Find Command Examples
- 15 Awesome Gmail Tips and Tricks
- 15 Awesome Google Search Tips and Tricks
- RAID 0, RAID 1, RAID 5, RAID 10 Explained with Diagrams
- Can You Top This? 15 Practical Linux Top Command Examples
- Top 5 Best System Monitoring Tools
- Top 5 Best Linux OS Distributions
- How To Monitor Remote Linux Host using Nagios 3.0
- Awk Introduction Tutorial – 7 Awk Print Examples
- How to Backup Linux? 15 rsync Command Examples
- The Ultimate Wget Download Guide With 15 Awesome Examples
- Top 5 Best Linux Text Editors
- Packet Analyzer: 15 TCPDUMP Command Examples
- The Ultimate Bash Array Tutorial with 15 Examples
- 3 Steps to Perform SSH Login Without Password Using ssh-keygen & ssh-copy-id
- Unix Sed Tutorial: Advanced Sed Substitution Examples
- UNIX / Linux: 10 Netstat Command Examples
- The Ultimate Guide for Creating Strong Passwords
- 6 Steps to Secure Your Home Wireless Network
- Turbocharge PuTTY with 12 Powerful Add-Ons

CATEGORIES

- Linux Tutorials

- Vim Editor
- Sed Scripting
- Awk Scripting
- Bash Shell Scripting
- Nagios Monitoring
- OpenSSH
- IPTables Firewall
- Apache Web Server
- MySQL Database
- Perl Programming
- Google Tutorials
- Ubuntu Tutorials
- PostgreSQL DB
- Hello World Examples
- C Programming
- C++ Programming
- DELL Server Tutorials
- Oracle Database
- VMware Tutorials

About The Geek Stuff

Linux 101 Hacks Book My name is **Ramesh Natarajan**. I will be posting instruction guides, how-to, troubleshooting tips and tricks on Linux, database, hardware, security and web. My focus is to write articles that will either teach you or help you resolve a problem. Read more about Ramesh Natarajan and the blog.

Contact Us

Email Me : Use this Contact Form to get in touch me with your comments, questions or suggestions about this site. You can also simply drop me a line to say hello!.

Follow us on Google+

Follow us on Twitter

Become a fan on Facebook

Support Us

Support this blog by purchasing one of my ebooks.

Bash 101 Hacks eBook

Sed and Awk 101 Hacks eBook

Vim 101 Hacks eBook

Nagios Core 3 eBook