

# Estructura de un Archivo ELF: Una Guía Detallada

Un archivo ELF (Executable and Linkable Format) es el formato estándar para ejecutables, bibliotecas compartidas, archivos de objetos y volcados de memoria ( `core dumps` ) en sistemas Unix-like (como Linux). Es una estructura altamente organizada que permite al sistema operativo cargar y ejecutar programas, o al enlazador combinar diferentes piezas de código.

Un archivo ELF se compone de varias partes principales:

## 1. Cabecera ELF ( `ELF Header` )

Es lo primero en cualquier archivo ELF y describe la estructura general del archivo. Es como la **tarjeta de identificación** del archivo. Contiene campos cruciales para interpretar el resto del archivo.

Información clave para `ft_nm` :

- **Magic Number** ( `e_ident` ): Los primeros 4 bytes son `0x7F 'E' 'L' 'F'` , que identifican el archivo como ELF. ¡Ya lo estás comprobando!
- **Clase de Arquitectura** ( `e_ident[EI_CLASS]` ): Indica si el ELF es de 32 bits ( `ELFCLASS32` ) o 64 bits ( `ELFCLASS64` ). ¡Ya lo estás detectando con `find_bits` !
- **Endianness** ( `e_ident[EI_DATA]` ): Especifica el orden de bytes (little-endian o big-endian). ¡Ya lo estás detectando con `find_endianness` ! Es crucial para leer correctamente valores de múltiples bytes (direcciones, offsets, tamaños).
- **Versión ELF** ( `e_ident[EI_VERSION]` ): La versión del formato ELF.
- **Tipo de Objeto** ( `e_type` ):
  - `ET_REL` : Archivo reubicable (objeto, `.o` ).
  - `ET_EXEC` : Archivo ejecutable.
  - `ET_DYN` : Biblioteca compartida ( `.so` ) o ejecutable cargado dinámicamente.
  - `ET_CORE` : Volcado de memoria.
- **Arquitectura de la Máquina** ( `e_machine` ): Indica la arquitectura del procesador (ej. `EM_X86_64` para x86-64, `EM_386` para i386, `EM_AARCH64` para ARM64).
- **Versión** ( `e_version` ): Debe ser `EV_CURRENT` .
- **Dirección de Entrada** ( `e_entry` ): Para ejecutables, es la dirección de memoria virtual donde el sistema operativo debe comenzar la ejecución.
- **Offset de la Tabla de Cabeceras de Programa** ( `e_phoff` ): La posición dentro del archivo donde comienza la Tabla de Cabeceras de Programa.
- **Offset de la Tabla de Cabeceras de Sección** ( `e_shoff` ): La posición dentro del archivo donde comienza la Tabla de Cabeceras de Sección.

- **Flags ( `e_flags` )**: Flags específicos de la arquitectura.
  - **Tamaño de la Cabecera ELF ( `e_ehsize` )**: Tamaño de esta cabecera en bytes.
  - **Tamaño de las Entradas de la Cabecera de Programa ( `e_phentsize` )**: Tamaño de una entrada en la Tabla de Cabeceras de Programa.
  - **Número de Entradas de la Cabecera de Programa ( `e_phnum` )**: Número de entradas en la Tabla de Cabeceras de Programa.
  - **Tamaño de las Entradas de la Cabecera de Sección ( `e_shentsize` )**: Tamaño de una entrada en la Tabla de Cabeceras de Sección.
  - **Número de Entradas de la Cabecera de Sección ( `e_shnum` )**: Número de entradas en la Tabla de Cabeceras de Sección.
  - **Índice de la Sección de Nombres ( `e_shstrndx` )**: El índice de la entrada en la Tabla de Cabeceras de Sección que contiene los nombres de las secciones.
- 

## 2. Tabla de Cabeceras de Programa ( `Program Header Table` o `Segment Header Table` )

Esta tabla es crucial para el **cargador de programas** del sistema operativo. Describe cómo el archivo debe ser mapeado en la memoria en tiempo de ejecución. Cada entrada en esta tabla se llama **Program Header** o **Segment Descriptor**.

**Información clave para `ft_nm`**: (Aunque `nm` se enfoca más en las secciones, entender los segmentos es útil para el contexto).

- **Tipo de Segmento ( `p_type` )**: Indica el propósito del segmento (ej., `PT_LOAD` para segmentos cargables, `PT_DYNAMIC` para información de enlace dinámico).
  - **Offset de Archivo ( `p_offset` )**: Offset del segmento dentro del archivo ELF.
  - **Dirección Virtual ( `p_vaddr` )**: Dirección virtual donde se cargará el segmento en memoria.
  - **Tamaño en Archivo ( `p_filesz` )**: Tamaño del segmento en el archivo.
  - **Tamaño en Memoria ( `p_memsz` )**: Tamaño del segmento en memoria (puede ser mayor que `p_filesz` si incluye datos inicializados a cero).
  - **Flags ( `p_flags` )**: Permisos de lectura/escritura/ejecución ( `PF_R` , `PF_W` , `PF_X` ).
  - **Alineación ( `p_align` )**: Requisito de alineación del segmento en memoria.
- 

## 3. Tabla de Cabeceras de Sección ( `Section Header Table` )

Esta tabla describe las secciones del archivo. Las **secciones** son unidades más pequeñas y lógicas dentro de los segmentos, utilizadas por el enlazador y las herramientas de depuración. Aquí es donde `nm` busca la mayoría de su información.

## Información clave para `ft_nm`:

Cada entrada en esta tabla es un **Section Header** y contiene:

- **Nombre de la Sección ( `sh_name` )**: Un offset al string que contiene el nombre de la sección en la sección de strings de los nombres de sección ( `.shstrtab` ).
    - **Necesitas encontrar la sección `.shstrtab`** para resolver estos nombres.
  - **Tipo de Sección ( `sh_type` )**: El propósito de la sección (ej., `SHT_PROGBITS` para código o datos, `SHT_SYMTAB` para la tabla de símbolos, `SHT_STRTAB` para la tabla de strings).
  - **Flags ( `sh_flags` )**: Atributos de la sección (ej., `SHF_ALLOC` si se carga en memoria, `SHF_EXECINSTR` si contiene código ejecutable, `SHF_WRITE` si es escribible).
  - **Dirección Virtual ( `sh_addr` )**: Dirección virtual de la sección si se carga en memoria.
  - **Offset de Archivo ( `sh_offset` )**: La posición de la sección dentro del archivo ELF.
  - **Tamaño de la Sección ( `sh_size` )**: El tamaño de la sección en bytes.
  - **Enlace ( `sh_link` )**: Índice de otra sección relacionada (ej., para una tabla de símbolos, esto puede apuntar a su tabla de strings asociada).
  - **Información ( `sh_info` )**: Información adicional específica del tipo de sección.
  - **Alineación ( `sh_addralign` )**: Requisito de alineación de la sección en memoria.
  - **Tamaño de Entrada ( `sh_entsize` )**: Si la sección contiene entradas de tamaño fijo (como la tabla de símbolos), este es el tamaño de cada entrada.
- 

## 4. Secciones Importantes para `ft_nm`

Dentro de la Tabla de Cabeceras de Sección, hay algunas secciones específicas que `nm` necesita analizar:

- **`.symtab` (Symbol Table Section)**:
  - Contiene la **tabla de símbolos** del archivo. Cada entrada es una estructura `Elf32_Sym` o `Elf64_Sym`.
  - **Necesitas**:
    - Su `sh_offset` para saber dónde empieza en el archivo.
    - Su `sh_size` para saber cuántos bytes ocupa.
    - Su `sh_entsize` para saber el tamaño de cada entrada de símbolo.
    - Su `sh_link` para encontrar la tabla de strings de símbolos ( `.strtab` ) asociada.
- **`.strtab` (String Table Section)**:
  - Contiene las **cadenas de caracteres** (nombres) de los símbolos listados en `.symtab`. Los símbolos en `.symtab` solo almacenan un offset a esta sección para su nombre.

- **Necesitas:**

- Su `sh_offset` para saber dónde empieza.
- Su `sh_size` para saber su tamaño.

- **`.shstrtab` (Section Header String Table Section):**

- Contiene los **nombres de todas las secciones** del archivo. La cabecera ELF (`e_shstrndx`) apunta a esta sección.

- **Necesitas:**

- Su `sh_offset` para saber dónde empieza.
  - Su `sh_size` para saber su tamaño.
- 

## 5. Entradas de la Tabla de Símbolos ( `Symbol Table Entry` )

Cada entrada en `.symtab` es un **símbolo**. Aquí es donde obtendrás la información que `nm` imprime.

### Información clave para `ft_nm`:

Para cada `Elf32_Sym` o `Elf64_Sym` (la estructura depende de si el ELF es de 32 o 64 bits):

- **`st_name`**: Un offset a la cadena de caracteres que es el nombre del símbolo, dentro de la sección `.strtab` asociada.
  - **`st_value`**: El valor del símbolo. Puede ser una dirección, un tamaño, o cualquier otra cosa dependiendo del contexto. Para la mayoría de los símbolos, es la dirección del símbolo.
  - **`st_size`**: El tamaño del símbolo.
  - **`st_info`**: Contiene el **tipo** y el **binding** del símbolo.
    - `ELF32_ST_BIND(st_info)` / `ELF64_ST_BIND(st_info)`: Extrae el binding (ej., `STB_LOCAL`, `STB_GLOBAL`, `STB_WEAK`).
    - `ELF32_ST_TYPE(st_info)` / `ELF64_ST_TYPE(st_info)`: Extrae el tipo (ej., `STT_NOTYPE`, `STT_OBJECT`, `STT_FUNC`, `STT_SECTION`, `STT_FILE`).
  - **`st_other`**: Usado para visibilidad (ej., `STV_DEFAULT`, `STV_HIDDEN`).
  - **`st_shndx`**: El índice de la sección a la que se refiere el símbolo. Por ejemplo, si es una función, este es el índice de la sección de código donde reside. Puede ser un valor especial como `SHN_ABS` (valor absoluto), `SHN_COMMON` (símbolo común) o `SHN_UNDEF` (símbolo indefinido, es decir, externo).
-

# Guía Clara para Construir tu `ft_nm` (Flujo de Información)

Basado en lo que ya tienes y la información del ELF, aquí está la secuencia lógica para construir tu `nm` :

## 1. Lectura del Archivo y Mapeo en Memoria (¡Ya lo haces!):

- Abre el archivo.
- Obtén su tamaño ( `fstat` ).
- Mapea el archivo en memoria ( `mmap` ).
- Verifica la firma ELF ( `0x7F 'E' 'L' 'F'` ).
- Detecta la **clase (32/64 bits)** y el **endianness**. (¡Ya lo tienes!)

## 2. Parsear la Cabecera ELF ( `Elf32_Ehdr` o `Elf64_Ehdr` ):

- Dependiendo de `aux->bits`, castear `aux->file_content_ptr` a `Elf32_Ehdr *` o `Elf64_Ehdr *`. Almacenar esto en `aux->elf32_header` o `aux->elf64_header`.
- Esta cabecera te dará los offsets y números de entradas para las Tablas de Cabeceras de Programa y Sección.
- **Crucial:** El `e_shoff` (offset a la tabla de cabeceras de sección) y `e_shnum` (número de secciones) serán vitales. También `e_shstrndx` (índice de la sección `.shstrtab`).

## 3. Localizar la Tabla de Cabeceras de Sección:

- Usa `e_shoff` de la cabecera ELF para encontrar el inicio de la Tabla de Cabeceras de Sección en la memoria mapeada.
- Necesitarás un puntero a `Elf32_Shdr` o `Elf64_Shdr` (dependiendo de `aux->bits`).

## 4. Localizar la Sección de Nombres de Sección ( `.shstrtab` ):

- Usa el índice `e_shstrndx` de la Cabecera ELF para acceder a la entrada correspondiente en la Tabla de Cabeceras de Sección.
- De esa entrada (la que describe `.shstrtab`), obtén su `sh_offset` y `sh_size`.
- Esto te dará la dirección y tamaño de la sección que contiene los nombres de todas las demás secciones. Esto es esencial para **nombrar las secciones**.

## 5. Iterar a través de la Tabla de Cabeceras de Sección:

- Recorre las `e_shnum` entradas de la Tabla de Cabeceras de Sección.
- Para cada entrada, necesitas:
  - Obtener su **nombre** usando `sh_name` como offset en la sección `.shstrtab` que acabas de localizar.

- Buscar secciones con tipo `SHT_SYMTAB` y `SHT_STRTAB` (para la tabla de símbolos y su tabla de strings asociada, respectivamente).
- Registrar sus offsets, tamaños y enlaces( `sh_link` ).

## 6. Localizar la Tabla de Símbolos ( `.symtab` ) y su Tabla de Strings ( `.strtab` ):

- Una vez que has iterado por las cabeceras de sección, identifica la sección `SHT_SYMTAB` . Esta es tu tabla de símbolos.
- Usa su campo `sh_link` para encontrar el índice de la sección `SHT_STRTAB` asociada(su tabla de strings de símbolos).
- Obtén los offsets y tamaños de estas dos secciones.

## 7. Parsear las Entradas de la Tabla de Símbolos:

- Usa el `sh_offset` de `.symtab` y el `sh_entsize` de `.symtab` para iterar sobre cada `Elf32_Sym` o `Elf64_Sym` en la memoria mapeada.
- Para cada símbolo:
  - Obtén su **nombre** usando `st_name` como offset en la sección `.strtab` que acabas de localizar.
  - Extrae el **tipo de símbolo** y el **binding** de `st_info` (usando las macros `ELF32_ST_TYPE` , `ELF32_ST_BIND` o sus equivalentes para 64 bits).
  - Obtén su **valor** de `st_value` .
  - Identifica su **sección** usando `st_shndx` .

## 8. Determinar el Tipo de Símbolo (el carácter que muestra `nm` ):

- Esta es la parte más compleja. El carácter(T, t, D, d, B, b, U, W, etc.) depende de una combinación de:
  - `st_shndx` : Si es `SHN_UNDEF` (U), `SHN_ABS` (A), `SHN_COMMON` (C).
  - `st_type` : Si es `STT_FUNC` (T o t), `STT_OBJECT` (D, d, B, b), `STT_SECTION` (S o s).
  - `st_bind` : Si es `STB_GLOBAL` (mayúscula) o `STB_LOCAL` (minúscula), `STB_WEAK` (W o w).
  - **Flags de la sección referenciada**: Si la sección es escribible( `SHF_WRITE` ), ejecutable( `SHF_EXECINSTR` ), etc. Por ejemplo, `.text` es T, `.data` es D, `.bss` es B.
  - **Visibilidad ( `st_other` )**: Afecta si un símbolo débil es W o w .

## 9. Filtrar y Ordenar Símbolos (para las flags de `nm` ):

- Aplica las reglas de las flags que soportes( `-a` , `-g` , `-u` , `-r` , `-p` ).
- Ordena los símbolos. Por defecto, `nm` los ordena alfabéticamente. La flag `-r` los invierte.

## 10. Imprimir la Salida:

- Formatea la salida según las convenciones de `nm` (dirección, tipo de símbolo, nombre).
-