Important Bin Tools in Linux

Binary tools in Linux are essential for analyzing, debugging, and understanding executable files, object files, and shared libraries. These tools help inspect the internals of compiled binaries, including symbol tables, disassembly, ELF (Executable and Linkable Format) headers, and binary size information.

---

1. `nm` (Symbol Table)

Purpose:

The `nm` command is used to list symbols from object files or executables. It helps analyze function and variable names within a binary, their types, and memory addresses.

Key Features:

- Displays symbol tables in object files and executables.
- Identifies global and local symbols.
- Categorizes symbols by type (functions, variables, etc.).

Common Options:

- `nm a.out` → Lists symbols in `a.out`.
- `nm -g a.out` → Shows only external (global) symbols.
- `nm -C a.out` → Demangles C++ symbols.

Example:

```
gcc -c sample.c  # Compile without linking
nm sample.o      # Display symbols in object file
```

Sample Output:

```
00000000 T main
00000004 T func
00000000 D globalVar
00000000 B uninitializedVar
```

- `T` → Text (code) section (function).
- `D` → Initialized data section.
- `B` → Uninitialized data section (BSS).

---

2. `objdump` (Object File Information)

Purpose:

The `objdump` command provides detailed information about an object file or executable, including disassembly, section headers, and symbols.

Key Features:

- Extracts disassembly and sections.
- Analyzes object files and executables.
- Displays assembly code of a binary.

Common Options:

- `objdump -d a.out` → Disassemble executable.
- `objdump -S a.out` → Disassemble with source code (if available).
- `objdump -h a.out` → Show section headers.

Example:

```
gcc -g -o test test.c
objdump -d test
```

Sample Output:

```
0000000000001129 <main>:
    1129:   55                      push   %rbp
    112a:   48 89 e5                mov    %rsp,%rbp
    112d:   b8 00 00 00 00          mov    $0x0,%eax
    1132:   5d                      pop    %rbp
    1133:   c3                      ret
```

This disassembles the binary, showing assembly instructions for `main()`.

---

3. `readelf` (ELF File Information)

Purpose:

The `readelf` command extracts and displays information from ELF files, including headers, section tables, and program headers.

Key Features:

- Retrieves ELF header information.
- Examines section headers.
- Provides details about segments, symbols, and relocations.

Common Options:

- `readelf -h a.out` → Show ELF header.
- `readelf -S a.out` → Show section headers.
- `readelf -r a.out` → Show relocation entries.

Example:

```
gcc -o test test.c
readelf -h test
```

Sample Output:

```
ELF Header:
  Magic:   7f 45 4c 46
  Class:   ELF64
  Type:    EXEC (Executable file)
  Machine: x86-64
  Entry point address: 0x400400
```

The ELF header gives details about the binary format, architecture, and entry point.

---

4. `size` (Binary Size Information)

Purpose:

The `size` command displays the memory footprint of an executable or object file, including the size of text (code), data, and BSS sections.

Key Features:

- Determines binary size information.
- Displays memory consumption of different sections.

Common Options:

- `size a.out` → Show section sizes of the binary.
- `size -B a.out` → Display sizes in bytes.

Example:

```
gcc -o test test.c
size test
```

Sample Output:

```
   text    data     bss     dec     hex filename
   1240     532      16    1788     6fc test
```

- `text` → Size of the code section.
- `data` → Size of initialized variables.

   - `bss` → Size of uninitialized variables.

Conclusion

These tools (`nm`, `objdump`, `readelf`, and `size`) are essential for analyzing ELF binaries in Linux. They help in debugging, reverse engineering, and optimizing compiled executables. By mastering them, developers can gain deeper insights into binary structures and execution.