

**NAME**

**ldd** – print shared object dependencies

**SYNOPSIS**

**ldd** [*option*]... *file*...

**DESCRIPTION**

**ldd** prints the shared objects (shared libraries) required by each program or shared object specified on the command line. An example of its use and output is the following:

```
$ ldd /bin/ls
linux-vdso.so.1 (0x00007ffcc3563000)
libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f87e5459000)
libcap.so.2 => /lib64/libcap.so.2 (0x00007f87e5254000)
libc.so.6 => /lib64/libc.so.6 (0x00007f87e4e92000)
libpcre.so.1 => /lib64/libpcre.so.1 (0x00007f87e4c22000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f87e4a1e000)
/lib64/ld-linux-x86-64.so.2 (0x00005574bf12e000)
libattr.so.1 => /lib64/libattr.so.1 (0x00007f87e4817000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f87e45fa000)
```

In the usual case, **ldd** invokes the standard dynamic linker (see **ld.so(8)**) with the **LD\_TRACE\_LOADED\_OBJECTS** environment variable set to 1. This causes the dynamic linker to inspect the program's dynamic dependencies, and find (according to the rules described in **ld.so(8)**) and load the objects that satisfy those dependencies. For each dependency, **ldd** displays the location of the matching object and the (hexadecimal) address at which it is loaded. (The *linux-vdso* and *ld-linux* shared dependencies are special; see **vdso(7)** and **ld.so(8)**.)

**Security**

Be aware that in some circumstances (e.g., where the program specifies an ELF interpreter other than *ld-linux.so*), some versions of **ldd** may attempt to obtain the dependency information by attempting to directly execute the program, which may lead to the execution of whatever code is defined in the program's ELF interpreter, and perhaps to execution of the program itself. (Before glibc 2.27, the upstream **ldd** implementation did this for example, although most distributions provided a modified version that did not.)

Thus, you should *never* employ **ldd** on an untrusted executable, since this may result in the execution of arbitrary code. A safer alternative when dealing with untrusted executables is:

```
$ objdump -p /path/to/program | grep NEEDED
```

Note, however, that this alternative shows only the direct dependencies of the executable, while **ldd** shows the entire dependency tree of the executable.

**OPTIONS****--version**

Print the version number of **ldd**.

**-v, --verbose**

Print all information, including, for example, symbol versioning information.

**-u, --unused**

Print unused direct dependencies. (Since glibc 2.3.4.)

**-d, --data-relocs**

Perform relocations and report any missing objects (ELF only).

**-r, --function-relocs**

Perform relocations for both data objects and functions, and report any missing objects or functions (ELF only).

**--help** Usage information.

**BUGS**

**ldd** does not work on a.out shared libraries.

**ldd** does not work with some extremely old a.out programs which were built before **ldd** support was added to the compiler releases. If you use **ldd** on one of these programs, the program will attempt to run with *argc* = 0 and the results will be unpredictable.

**SEE ALSO**

**pldd**(1), **sprof**(1), **ld.so**(8), **ldconfig**(8)