How to show all shared libraries used by executables in Linux?

Asked 14 years, 2 months ago Modified 7 months ago Viewed 403k times

272

I'd like to know which libraries are used by executables on my system. More specifically, I'd like to rank which libraries are used the most, along with the binaries that use them. How can I do this?

linux shared-libraries

Share Improve this question Follow

edited Nov 29, 2017 at 14:27

Ciro Santilli OurBigBook.com
321k 91 1138 923

asked Sep 8, 2008 at 17:02

Alan Szlosek
3,071 3 19 12

You will probably not be able to get an exact number if the executables use `dlopen` . – jxh Feb 2, 2018 at 19:16

13 Answers

Sorted by: Highest score (default)

314

1. Use `ldd` to list shared libraries for each executable.
2. Cleanup the output
3. Sort, compute counts, sort by count

To find the answer for all executables in the "/bin" directory:

```
find /bin -type f -perm /a+x -exec ldd {} \; \
| grep so \
| sed -e '/^[^\t]/ d' \
| sed -e 's/\t//' \
| sed -e 's/.*=..//' \
| sed -e 's/ (0.*)//' \
| sort \
| uniq -c \
| sort -n
```

Change "/bin" above to "/" to search all directories.

Output (for just the /bin directory) will look something like this:

```
  1 /lib64/libnsl.so.1
  1 /lib64/libpcre.so.0
  1 /lib64/libproc-3.2.7.so
  1 /usr/lib64/libbeecrypt.so.6
  1 /usr/lib64/libbz2.so.1
  1 /usr/lib64/libelf.so.1
  1 /usr/lib64/libpopt.so.0
  1 /usr/lib64/librpm-4.4.so
  1 /usr/lib64/librpmdb-4.4.so
  1 /usr/lib64/librpmio-4.4.so
  1 /usr/lib64/libsqlite3.so.0
  1 /usr/lib64/libstdc++.so.6
  1 /usr/lib64/libz.so.1
  2 /lib64/libasound.so.2
  2 /lib64/libblkid.so.1
  2 /lib64/libdevmapper.so.1.02
  2 /lib64/libpam_misc.so.0
  2 /lib64/libpam.so.0
  2 /lib64/libuuid.so.1
  3 /lib64/libaudit.so.0
  3 /lib64/libcrypt.so.1
  3 /lib64/libdbus-1.so.3
  4 /lib64/libresolv.so.2
  4 /lib64/libtermcap.so.2
  5 /lib64/libacl.so.1
  5 /lib64/libattr.so.1
  5 /lib64/libcap.so.1
  6 /lib64/librt.so.1
  7 /lib64/libm.so.6
  9 /lib64/libpthread.so.0
 13 /lib64/libselinux.so.1
 13 /lib64/libsepol.so.1
 22 /lib64/libdl.so.2
 83 /lib64/ld-linux-x86-64.so.2
 83 /lib64/libc.so.6
```

Edit - Removed "grep -P"

Share Improve this answer Follow

edited May 29, 2012 at 19:52

Dave Jarvis
29.6k 38 176 307

answered Sep 8, 2008 at 17:21

John Vasileff
5,223 2 22 16

2 This is a great answer (I've up-voted it) but can you explain the "grep -P '\t.*so'" command? According to man, this interprets the pattern as a perl regexp, but my version of grep doesn't support it (man indicates this is a general issue). What bit of the regexp is perl-specific? – Bobby Jack Sep 8, 2008 at 17:36

2 I think you may need to use `ldd -v` – MountainX Apr 25, 2012 at 3:30

70 Be aware that `ldd` actually runs the executable with a special environment variable, and the Linux dynamic linker recognizes this flag and just outputs the libraries rather than running the executable. Look at the source to `ldd` ; on my system, it's a bash script. If the executable is statically linked and uses syscalls, and specifies a different loader, it can do arbitrary evil things. So don't use `ldd` on an executable you don't trust. – Barry Kelly Sep 25, 2013 at 15:26

2 'ldd' doesn't work for me on cross-compiled binaries. The question is about finding the libraries used by

programs for a different system ('readelf' mentioned in another answer, worked for me) – Tim Bird Dec 21, 2019 at 2:29

This is a big empty output in my case : / – Balázs Börcsök Nov 5 at 20:41

85

I didn't have ldd on my ARM toolchain so I used objdump:

$(CROSS_COMPILE)objdump -p

For instance:

```
objdump -p /usr/bin/python:

Dynamic Section:
  NEEDED               libpthread.so.0
  NEEDED               libdl.so.2
  NEEDED               libutil.so.1
  NEEDED               libssl.so.1.0.0
  NEEDED               libcrypto.so.1.0.0
  NEEDED               libz.so.1
  NEEDED               libm.so.6
  NEEDED               libc.so.6
  INIT                 0x0000000000416a98
  FINI                 0x000000000053c058
  GNU_HASH             0x0000000000400298
  STRTAB               0x000000000040c858
  SYMTAB               0x0000000000402aa8
  STRSZ                0x0000000000006cdb
  SYMENT               0x0000000000000018
  DEBUG                0x0000000000000000
  PLTGOT               0x0000000000832fe8
  PLTRELSZ             0x0000000000002688
  PLTREL               0x0000000000000007
  JMPREL               0x0000000000414410
  RELA                 0x0000000000414398
  RELASZ               0x0000000000000078
  RELAENT              0x0000000000000018
  VERNEED              0x0000000000414258
  VERNEEDNUM           0x0000000000000008
  VERSYM               0x0000000000413534
```

Share Improve this answer Follow

answered Mar 20, 2013 at 10:28

smichak

4,606 3 34 47

2 This should be safe too, unlike `ldd` which shouldn't be used on untrusted executables. – PSkocik Sep 2, 2017 at 18:16

1 Also, `objdump -p` shows additional information like the `RPATH`, which may be of help when investigating dynamic linking issues with your executable. – sitaktif Oct 4, 2018 at 10:27

ldd and objdump -p have different outputs (ldd outputs more libs) – ychaouche Jan 30 at 12:17

62

On Linux I use:

```
lsof -P -T -p Application_PID
```

This works better than `ldd` when the executable uses a non default loader

Share Improve this answer Follow

edited May 23, 2017 at 12:34
Community Bot
1 1

answered Nov 17, 2011 at 23:54
Fabiano Tarlao
2,953 35 39

Used this to find out if mariadb was actually using tc-malloc, which gets loaded by LD_PRELOAD. Works great. – cmc Feb 25, 2013 at 16:33

2 I was looking for something that would show me '.so' for a given pid. This is exactly what I needed. Thanks! – Leo Ufimtsev Aug 11, 2017 at 20:27

ldd and objdump -p have different outputs (ldd outputs more libs) – ychaouche Dec 30, 2021 at 13:51

@ychaouche in this old answer I pointed out that lsof is better than ldd, in specific situations, I never mentioned objdump. Am I missing something? – Fabiano Tarlao Jan 5 at 15:54

@FabianoTarlao, oh sorry, I've added my comment to the wrong answere ! the comment was for this answere stackoverflow.com/a/15520982/212044 – ychaouche Jan 30 at 12:18

57

to learn what libraries a binary uses, use ldd

```
ldd path/to/the/tool
```

You'd have to write a little shell script to get to your system-wide breakdown.

Share Improve this answer Follow

answered Sep 8, 2008 at 17:04
pilif
12.4k 5 34 31

19

Check shared library dependencies of a program executable

To find out what libraries a particular executable depends on, you can use ldd command. This command invokes dynamic linker to find out library dependencies of an executable.

Note that it is NOT recommended to run ldd with any untrusted third-party executable because some versions of ldd may directly invoke the executable to identify its library dependencies, which can be security risk.

Instead, a safer way to show library dependencies of an unknown application binary is to use the following command.

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

for more info

Share Improve this answer Follow

answered Apr 24, 2015 at 4:53

kayle
1,096 12 20

17

`readelf -d` recursion

`redelf -d` produces similar output to `objdump -p` which was mentioned at:
https://stackoverflow.com/a/15520982/895245

But beware that dynamic libraries can depend on other dynamic libraries, to you have to recurse.

Example:

```
readelf -d /bin/ls | grep 'NEEDED'
```

Sample ouptut:

```
0x0000000000000001 (NEEDED)             Shared library: [libselinux.so.1]
0x0000000000000001 (NEEDED)             Shared library: [libacl.so.1]
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

Then:

```
$ locate libselinux.so.1
/lib/i386-linux-gnu/libselinux.so.1
/lib/x86_64-linux-gnu/libselinux.so.1
/mnt/debootstrap/lib/x86_64-linux-gnu/libselinux.so.1
```

Choose one, and repeat:

Sample output:

```
0x0000000000000001 (NEEDED)             Shared library: [libpcre.so.3]
0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
0x0000000000000001 (NEEDED)             Shared library: [ld-linux-x86-64.so.2]
```

And so on.

`/proc/<pid>/maps` for running processes

This is useful to find all the libraries currently being used by running executables. E.g.:

```
sudo awk '/\.so/{print $6}' /proc/1/maps | sort -u
```

shows all currently loaded dynamic dependencies of `init` (PID `1`):

```
/lib/x86_64-linux-gnu/ld-2.23.so
/lib/x86_64-linux-gnu/libapparmor.so.1.4.0
/lib/x86_64-linux-gnu/libaudit.so.1.0.0
/lib/x86_64-linux-gnu/libblkid.so.1.1.0
/lib/x86_64-linux-gnu/libc-2.23.so
/lib/x86_64-linux-gnu/libcap.so.2.24
/lib/x86_64-linux-gnu/libdl-2.23.so
/lib/x86_64-linux-gnu/libkmod.so.2.3.0
/lib/x86_64-linux-gnu/libmount.so.1.1.0
/lib/x86_64-linux-gnu/libpam.so.0.83.1
/lib/x86_64-linux-gnu/libpcre.so.3.13.2
/lib/x86_64-linux-gnu/libpthread-2.23.so
/lib/x86_64-linux-gnu/librt-2.23.so
/lib/x86_64-linux-gnu/libseccomp.so.2.2.3
/lib/x86_64-linux-gnu/libselinux.so.1
/lib/x86_64-linux-gnu/libuuid.so.1.3.0
```

This method also shows libraries opened with `dlopen`, tested with this minimal setup hacked up with a `sleep(1000)` on Ubuntu 18.04.

See also: https://superuser.com/questions/310199/see-currently-loaded-shared-objects-in-linux/1243089

Share Improve this answer Follow

edited Sep 13, 2018 at 21:24

answered Mar 26, 2017 at 8:57

Ciro Santilli
OurBigBook.com
321k 91 1138 923

On OS X by default there is no `ldd`, `objdump` or `lsof`. As an alternative, try `otool -L`:

```
$ otool -L `which openssl`
/usr/bin/openssl:
    /usr/lib/libcrypto.0.9.8.dylib (compatibility version 0.9.8, current version 0.9.8)
    /usr/lib/libssl.0.9.8.dylib (compatibility version 0.9.8, current version 0.9.8)
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1213.0.0)
```

In this example, using `which openssl` fills in the fully qualified path for the given executable and current user environment.

Share Improve this answer Follow

answered Jan 27, 2015 at 0:29

bluebadge
1,179 1 8 5

7

On UNIX system, suppose binary (executable) name is test. Then we use the following command to list the libraries used in the test is

```
ldd test
```

Share Improve this answer Follow

edited Apr 20, 2011 at 11:41

Cody Gray ♦
235k 50 483 563

answered Apr 20, 2011 at 8:07

Raghwendra
71 1 1

4

With `ldd` you can get the libraries that tools use. To rank the usage of libraries for a set of tool you can use something like the following command.

```
ldd /bin/* /usr/bin/* ... | sed -e '/^[^\t]/ d; s/^\t\(.* => \)\?\([^ ]*\) (.*/\2/g' | sort | uniq -c
```

(Here `sed` strips all lines that do not start with a tab and the filters out only the actual libraries. With `sort | uniq -c` you get each library with a count indicating the number of times it occurred.)

You might want to add `sort -g` at the end to get the libraries in order of usage.

Note that you probably get lines two non-library lines with the above command. One of static executables ("not a dynamic executable") and one without any library. The latter is the result of `linux-gate.so.1` which is not a library in your file system but one "supplied" by the kernel.

Share Improve this answer Follow

answered Sep 8, 2008 at 17:35

mweerden
13.2k 5 30 31

2

```
/proc/<pid>/maps
```

For example is the process id is 2601 then the command is

```
cat /proc/2601/maps
```

And the output is like

```
7fb37a8f2000-7fb37a8f4000 r-xp 00000000 08:06 4065647                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/network_networkmanager.so
7fb37a8f4000-7fb37aaf3000 ---p 00002000 08:06 4065647                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/network_networkmanager.so
7fb37aaf3000-7fb37aaf4000 r--p 00001000 08:06 4065647                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/network_networkmanager.so
7fb37aaf4000-7fb37aaf5000 rw-p 00002000 08:06 4065647                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/network_networkmanager.so
7fb37aaf5000-7fb37aafe000 r-xp 00000000 08:06 4065646                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/config_gnome3.so
7fb37aafe000-7fb37acfd000 ---p 00009000 08:06 4065646                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/config_gnome3.so
7fb37acfd000-7fb37acfe000 r--p 00008000 08:06 4065646                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/config_gnome3.so
7fb37acfe000-7fb37acff000 rw-p 00009000 08:06 4065646                    /usr/lib/x86_64-
linux-gnu/libproxy/0.4.15/modules/config_gnome3.so
7fb37acff000-7fb37ad1d000 r-xp 00000000 08:06 3416761                    /usr/lib/x86_64-
linux-gnu/libproxy.so.1.0.0
7fb37ad1d000-7fb37af1d000 ---p 0001e000 08:06 3416761                    /usr/lib/x86_64-
linux-gnu/libproxy.so.1.0.0
7fb37af1d000-7fb37af1e000 r--p 0001e000 08:06 3416761                    /usr/lib/x86_64-
linux-gnu/libproxy.so.1.0.0
7fb37af1e000-7fb37af1f000 rw-p 0001f000 08:06 3416761                    /usr/lib/x86_64-
linux-gnu/libproxy.so.1.0.0
7fb37af1f000-7fb37af21000 r-xp 00000000 08:06 4065186                    /usr/lib/x86_64-
linux-gnu/gio/modules/libgiolibproxy.so
7fb37af21000-7fb37b121000 ---p 00002000 08:06 4065186                    /usr/lib/x86_64-
linux-gnu/gio/modules/libgiolibproxy.so
7fb37b121000-7fb37b122000 r--p 00002000 08:06 4065186                    /usr/lib/x86_64-
linux-gnu/gio/modules/libgiolibproxy.so
7fb37b122000-7fb37b123000 rw-p 00003000 08:06 4065186                    /usr/lib/x86_64-
linux-gnu/gio/modules/libgiolibproxy.so
```

Share Improve this answer Follow

answered Nov 30, 2018 at 9:43

SoSen
169 1 9

2

on ubuntu print packages related to an executable

```
ldd executable_name|awk '{print $3}'|xargs dpkg -S |awk -F  ":"  '{print $1}'
```


eyllanesc
227k 18 139 209


Shimon Doodkin
4,132 33 37

2

If you don't care about the path to the executable file -

```
ldd `which <executable>` # back quotes, not single quotes
```

Share Improve this answer Follow

answered Sep 2, 2020 at 5:07


nomad
411 5 11

0

I found this post very helpful as I needed to investigate dependencies from a 3rd party supplied library (32 vs 64 bit execution path(s)).

I put together a Q&D recursing bash script based on the 'readelf -d' suggestion on a RHEL 6 distro.

It is very basic and will test every dependency every time even if it might have been tested before (i.e very verbose). Output is very basic too.

```
#! /bin/bash

recurse ()
# Param 1 is the nuumber of spaces that the output will be prepended with
# Param 2 full path to library
{
#Use 'readelf -d' to find dependencies
dependencies=$(readelf -d ${2} | grep NEEDED | awk '{ print $5 }' | tr -d '[]')
for d in $dependencies; do
   echo "${1}${d}"
   nm=${d##*/}
   #libstdc++ hack for the '+'-s
   nm1=${nm//"+"/"\+"}
   # /lib /lib64 /usr/lib and /usr/lib are searched
   children=$(locate ${d} | grep -E "(^/(lib|lib64|usr/lib|usr/lib64)/${nm1})")
   rc=$?
   #at least locate... didn't fail
   if [ ${rc} == "0" ] ; then
      #we have at least one dependency
      if [ ${#children[@]} -gt 0 ]; then
         #check the dependeny's dependencies
         for c in $children; do
          recurse "  ${1}" ${c}
         done
      else
         echo "${1}no children found"
      fi
   else
```

```
# Q&D -- recurse needs 2 params could/should be supplied from cmdline
recurse "" !!full path to library you want to investigate!!
```

redirect the output to a file and grep for 'found' or 'failed'

Use and modify, at your own risk of course, as you wish.

Share Improve this answer Follow

answered Oct 19, 2017 at 14:20

Anders Domeij

1 1

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.