pylddwrap Documentation

Release 1.2.2

Selim Naji, Adam Radomski and Marko Ristin

Nov 04, 2022

Contents:

CHAPTER 1

lddwrap

Wrap ldd *nix utility to determine shared libraries required by a program.

class lddwrap.**Dependency**(*found*, *soname=None*, *path=None*, *mem_address=None*, *unused=None*)

Represent a shared library required by a program.

Variables

- **found** (*bool*) – True if ldd could resolve the library
- **soname** (*Optional[str]*) – library name
- **path** (*Optional[pathlib.Path]*) – path to the library
- **mem_address** (*Optional[str]*) – hex memory location
- **unused** (*Optional[bool]*) – library not used

Establishes

- ```
  not (self.mem_address is None and self.found) or (
  self.soname is not None and self.path is not None)
  ```
- ```
  not (self.path is None and self.found) or (
  self.soname is not None and self.mem_address is not None)
  ```
- ```
  not (self.soname is None and self.found) or (
  self.path is not None and self.mem_address is not None)
  ```

as_mapping()

Transform the dependency to a mapping.

Can be converted to JSON and similar formats.

Return type Mapping[str, Any]

lddwrap.**list_dependencies**(*path*, *unused=False*, *env=None*)

Retrieve a list of dependencies of the given binary.
  ```

```
>>> import shutil
>>> ls = shutil.which("ls")
>>> path = pathlib.Path(ls)
>>> deps = list_dependencies(path=path)
>>> deps[0].soname
'linux-vdso.so.1'
```

Parameters

- **path** (Path) – path to a file
- **unused** (bool) – if set, check if dependencies are actually used by the program
- **env** (Optional[Dict[str, str]]) – the environment to use.

  If env is None, currently active env will be used. Otherwise specified env is used.

Return type List[*Dependency*]

Returns list of dependencies

Requires

- path.is_file()

CHAPTER 2

README

2.1 pylddwrap

Pylddwrap wraps ldd *nix utility to determine shared libraries required by a program.

We need to dynamically package subset of our system at deployment time. Consequently, we have to determine the dependencies on shared libraries of our binaries programmatically.

The output of ldd Linux command, while informative, is not structured enough to be easily integrated into a program. At the time of this writing, we only found two alternative ldd wrappers on Internet python-ldd and ldd.py, but their output was either too basic for our use case or the project was still incipient.

Pylddwrap, in contrast, returns a well-structured list of the dependencies. The command-line tool outputs the dependencies either as a table (for visual inspection) or as a JSON-formatted string (for use with other tools). The included Python module lddwrap returns a Python object with type annotations so that it can be used readily by the deployment scripts and other modules.

For more information on the ldd tool, please see ldd manual.

2.2 Usage

2.2.1 Command-Line Tool pylddwrap

- Assume we need the dependencies of the /bin/ls. The following command gives them as a table:

```
pylddwrap /bin/ls
```

- The output of the command looks like this:

```
soname          | path                                    | found | mem_address        |
↪| unused
----------------+-----------------------------------------+-------+--------------------
↪+-------
linux-vdso.so.1 | None                                    | True  | 0x00007ffd8750f000|
↪| False
libselinux.so.1 | /lib/x86_64-linux-gnu/libselinux.so.1 | True  | 0x00007f4e73dc3000|
↪| True
libc.so.6       | /lib/x86_64-linux-gnu/libc.so.6       | True  | 0x00007f4e739f9000|
↪| False
libpcre.so.3    | /lib/x86_64-linux-gnu/libpcre.so.3    | True  | 0x00007f4e73789000|
↪| False
libdl.so.2      | /lib/x86_64-linux-gnu/libdl.so.2      | True  | 0x00007f4e73585000|
↪| False
None            | /lib64/ld-linux-x86-64.so.2           | True  | 0x00007f4e73fe5000|
↪| False
libpthread.so.0 | /lib/x86_64-linux-gnu/libpthread.so.0 | True  | 0x00007f4e73368000|
↪| False
```

- To obtain the dependencies as JSON, invoke:

```
pylddwrap --format json /bin/ls
```

- The JSON output is structured like this:

```
[
  {
    "soname": "linux-vdso.so.1",
    "path": "None",
    "found": true,
    "mem_address": "0x00007ffed857f000",
    "unused": false
  },
  ...
]
```

- You can also sort the table with `--sorted` which will sort by `soname`:

```
pylddwrap /bin/pwd --sorted
```

- Pylddwrap gives the table sorted by `soname`:

```
soname          | path                              | found | mem_address        |
↪unused
----------------+-----------------------------------+-------+--------------------+------
↪-
None            | /lib64/ld-linux-x86-64.so.2       | True  | 0x00007fd54894d000 | False
```

(continues on next page)

(continued from previous page)

```
libc.so.6       | /lib/x86_64-linux-gnu/libc.so.6 | True  | 0x00007fd548353000 | False
linux-vdso.so.1 | None                            | True  | 0x00007ffe0953f000 | False
```

Alternatively, you can sort by any other column. For example, to sort by `path`:

```
pylddwrap /bin/pwd --sorted path
```

- The output will be:

```
soname          | path                            | found | mem_address        |␣
↪unused
----------------+---------------------------------+-------+--------------------+------
↪-
linux-vdso.so.1 | None                            | True  | 0x00007ffe0953f000 | False
libc.so.6       | /lib/x86_64-linux-gnu/libc.so.6 | True  | 0x00007fd548353000 | False
None            | /lib64/ld-linux-x86-64.so.2     | True  | 0x00007fd54894d000 | False
```

2.2.2 ldwrap Python Module

We provide lddwrap Python module which you can integrate into your deployment scripts and other modules.

- The following example shows how to list the dependencies of /bin/ls:

```
import pathlib
import lddwrap

path = pathlib.Path("/bin/ls")
deps = lddwrap.list_dependencies(path=path)
for dep in deps:
    print(dep)

"""
soname: linux-vdso.so.1, path: None, found: True, mem_address: (0x00007ffe8e2fb000),␣
↪unused: None
soname: libselinux.so.1, path: /lib/x86_64-linux-gnu/libselinux.so.1, found: True,␣
↪mem_address: (0x00007f7759ccc000), unused: None
soname: libc.so.6, path: /lib/x86_64-linux-gnu/libc.so.6, found: True, mem_address:␣
↪(0x00007f7759902000), unused: None
...
"""
```

- List all dependencies of the /bin/ls utility and check if the direct dependencies are used. If unused for list_dependencies is set to False then the unused variable of the dependencies will not be determined and are therefore unknown and set to None. Otherwise information about direct usage will be retrieved and added to the dependencies.

```
import pathlib
import lddwrap

path = pathlib.Path("/bin/ls")
deps = lddwrap.list_dependencies(path=path, unused=True)
print(deps[1])
# soname: libselinux.so.1,
# path: /lib/x86_64-linux-gnu/libselinux.so.1,
# found: True,
```

(continues on next page)

(continued from previous page)

```
# mem_address: (0x00007f5a6064a000),
# unused: True
```

- Lddwrap operates normally with the environment variables of the caller. In cases where your dependencies are determined differently than the current environment, you pass a separate environment (in form of a dictionary) as an argument:

```
import os
import pathlib
import lddwrap

env = os.environ.copy()
env['LD_LIBRARY_PATH'] = "some/important/path"
path = pathlib.Path("/bin/ls")
deps = lddwrap.list_dependencies(path=path, env=env)
```

2.3 Installation

- Install pylddwrap with pip:

```
pip3 install pylddwrap
```

2.4 Development

- Check out the repository.
- In the repository root, create the virtual environment:

```
python3 -m venv venv3
```

- Activate the virtual environment:

```
source venv3/bin/activate
```

- Install the development dependencies:

```
pip3 install -e .[dev]
```

- Tests can be run directly using `unittest`:

```
python3 -m unittest discover tests/
```

2.4.1 Pre-commit Checks

We provide a set of pre-commit checks that lint and check code for formatting.

Namely, we use:

- yapf to check the formatting.
- The style of the docstrings is checked with pydocstyle.

- Static type analysis is performed with mypy.
- Various linter checks are done with pylint.

Apply the automatic formatting by running the `format` environment:

```
tox -e format
```

Run the pre-commit checks and tests using `tox`:

```
tox
```

2.5 Versioning

We follow Semantic Versioning. The version X.Y.Z indicates:

- X is the major version (backward-incompatible),
- Y is the minor version (backward-compatible), and
- Z is the patch version (backward-compatible bug fix).

CHAPTER 3

CHANGELOG

3.1 1.2.2

- Added support for paths with spaces (#26)

3.2 1.2.1

- Removed support for Python 3.5 (#23)
- Improved support of statically linked libraries (#22)

3.3 1.2.0

- Ignore not-indented ldd output (#14)
- Added handling of static libraries (#13)

3.4 1.1.0

- Added `--sorted` command-line option
- Refactored for easier and more thorough testing

3.5 1.0.2

- Added support for Python 3.7 and 3.8

3.6 1.0.1

- Fixed license badge in the Readme

3.7 1.0.0

- Initial version

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index