RepoLib Documentation

Ian Santopietro

May 08, 2019

Developer Documentation

RepoLib is a Python library and CLI tool-set for managing your software system software repositories. It's currently set up to handle APT repositories on Debian-based linux distributions.

RepoLib is intended to operate on DEB822-format sources. It aims to provide feature parity with software-properties for most commonly used functions. It also allows for simple, automated conversion from legacy "one-line" style source to newer DEB822 format sources. These sources will eventually deprecate the older one-line sources and are much easier to parse for both human and machine. For a detailed explanation of the DEB822 Source format, see *Explanation of the DEB822 Source Format*.

RepoLib provides much faster access to a subset of `SoftwareProperties` features than `SoftwareProperties` itself does. Its scope is somewhat more limited because of this, but the performance improvements gained are substantial. `SoftwareProperties` also does not yet feature support for managing DEB822 format sources, and instead only manages one-line sources.

RepoLib is available under a BSD 2-Clause License. Full License below:

CHAPTER 1

RepoLib

The *RepoLib* module simplifies working with APT sources, especially those stored in the DEB822 format. It translates these sources into Python Objects for easy reading, parsing, and manipulation of them within Python programs. The program takes a user-defined sources filename and reads the contents, parsing it into a Python object which can then be iterated upon and which uses native data types where practicable. The *RepoLib* module also allows easy creation of new DEB822-style sources from user-supplied data.

1.1 Source object

class repolib.Source (name='',enabled=True,types=[],uris=[],suites=[],components=[],options={},filename='example.source')

Create a new *Source object*. All parameters should be passed as keyword arguments. Each parameter has its own more detailed description below, but in short they are:

- *name* - The human-readable name of the source. (default: '')
- *enabled* - Whether the source is enabled or not at creation. (default: True)
- *types* - A list of the types that the source should use. (default: [])
- *uris* - A list of URIs from which to fetch software or check for updates. (default: [])
- *suites* - Suites in which to narrow available software. (default: [])
- *components* - Components of the source repository to enable. (default: [])
- *options* - Optional items affecting the source. (default: {})
- *filename* - The filename to save the source to on disk. (default: 'example.sources')

The following decribe how each of these are used.

1.1.1 name

This is a human-readable and nicely-formatted name to help a user recognize what this source is. Any unicode character is allowed in this field. If a source is opened which doesn't have a name field, the filename will be used.

name is a string value, set to `''` by default. If there is no name in a loaded source, it will be set to the same as the filename (minus the extension).

This field maps to the `X-Repolib-Name:` field in the .sources file, which is ignored by Apt and other standards-compliant sources parsers.

1.1.2 enabled

Apt sources can be disbaled without removing them entirely from the system. A disabled source is excluded from updates and new software installs, but it can be easily re-enabled at any time. It defaults to `True`.

This field maps to the `Enabled:` field in the .sources file. This is optional per the DEB822 standard, however if set to anything other than `no`, the source is considered enabled.

1.1.3 types

Debian archives may contain either binary packages or source code packages, and this value specifies which of those Apt should look for in the source. `deb` is used to look for binary packages, while `deb-src` looks for source code packages. RepoLib stores this value as a list of aptsourcetype-enum‘s, and defaults to ‘‘[AptSourceType.BINARY]‘.

This field maps to the `Types:` field in the sources file.

1.1.4 uris

A list of string values describing the URIs from which to fetch package lists and software for updates and installs. The currently recognized URI types are:

- file
- cdrom
- http
- ftp
- copy
- rsh
- ssh

Note: Although these are the currently-recognized official URI types, Apt can be extended with additional URI schemes through extension packages. Thus it is **not** recommended to parse URIs by type and instead rely on user input being correct and to throw exceptions when that isn’t the case.

1.1.5 suites

The Suite, also known as the **distribution** specifies a subdirectory of the main archive root in which to look for software. This is typically used to differentiate versions for the same OS, e.g. `disco` or `cosmic` for Ubuntu, or `squeeze` and `unstable` for Debian.

This value maps to the `Suites:` field in the sources file.

1.1.6 components

This value is a list of strings describing the enabled distro components to download software from. Common values include `main`, `restricted`, `nonfree`, etc.

1.1.7 options

This is a dictionary containing key value pairs of options to add to the source. Options often are used to restrict a source to certain CPU architectures or languages. Valid options include:

- `Architectures`
- `Languages`
- `Targets`
- `PDiffs`
- `By-Hash`

1.1.8 filename

This is a string value describing the filename to save the source to when using the save-to-disk-method. It defaults to `example.sources`

1.2 Methods

1.2.1 make_source_string()

Source.make_source_string() Takes the data from the *Source object* and makes it a printable string for output to console or for saving to a file. The *save_to_disk()* method uses this method as a basis for its file output.

Note: The `Name:` field output by this method is not suitable for directly saving to a file without additional processing. When using this method to generate data for manually saving to disk, be sure to replace `Name:` with `X-Repolib-Name:` first.

1.2.2 save_to_disk()

Source.save_to_disk() Takes all of the current data saved in the *Source object* and writes it to the disk. It uses the current *filename* attribute as the storage location within `/etc/apt/sources.list.d`.

1.2.3 load_from_file()

Source.load_from_file(filename=None) Loads the source from a file on disk. The location loaded depends on the *filename* parameter's value, as described below:

1.2.3.1 filename

The filename of the sources file to load from the disk. If ommitted, the method instead loads from the current *filename* attribute, otherwise the method sets the *filename* attribute to the value of this argument. For example:

```
>>> source = Source()
>>> source.filename
'example.sources'
>>> source.load_from_file(filename='google-chrome.sources')
>>> source.filename
'google-chrome.sources'
>>> source_with_name = Source()
>>> source_with_name.filename = 'ppa-system76-pop.sources'
>>> source_with_name.load_from_file()
>>> source_with_name.filename
'ppa-system76-pop.sources'
```

1.2.4 set_source_enabled()

Source.set_source_enabled(is_enabled) This method can be used to quickly set the *Source object*.‘‘types‘‘ attribute. Since the `types` attribute is a list of *AptSourceType* values, this method can quickly set the type to either of these values without needing to use the Enum directly. The argument *is_enabled* is a boolean value.

1.2.4.1 is_enabled

If `True`, the *Source object* `types` attribute is set to [*AptSourceType*.BINARY, *AptSourceType*.SOURCE]. Otherwise, it’s set to [*AptSourceType*.BINARY] only.

1.3 Source() Subclasses

There are a variety of subclasses of the *RepoLib* :ref‘source-object‘ class which tailor its functionality toward dealing with specific types of parent sources. The default :ref‘source-object‘ is best for dealing with native DEB822 sources on disk or as entered by user input.

1.3.1 SystemSource()

class repolib.SystemSource() A special *Source object* with its `filename` attribute pre-set to the path of the system software sources file. It is otherwise identical to the main *Source object* class.

1.3.2 DebLine()

class repolib.DebLine(line) A *Source object* class specifically intended to parse one-line debian repository list entries. This can be useful for both converting legacy sources to the new format, as well as a simple way for a user to add a new software source, since this format can be copy-pasted as a full line.

- *line* - The one-line source entry to parse into DEB822 format.

1.3.2.1 line

This is the one-line repository entry to operate on. Because this line contains the necessary information to fill in the *Source object* data completely, it is the only argument that the *DebLine()* accepts.

The line parameter is a string and *must* follow this format:

```
deb|deb-src ([options,...]) uri suite components
```

1.3.3 PPALine()

class repolib.PPALine(line) A *Source object* with convenience features for adding or removing PPA sources from launchpad.net. This class will exapand a `ppa:owner/source` format entry into a complete repository, and will download and save the signing key for the repository automatically (if an internet connection is available). It will also automatically name the entry based on data available on Launchpad.

**line* - The `ppa:` formatted entry to add.

1.3.3.1 line

A repository to add in the format:

```
ppa:onwer-name/repository-name
```

This will be automatically expanded to the uri ‘‘http://ppa.launchpad.net/owner-name/repository-name/ubuntu‘ and will be added with the current OS version codename serving as the “Suite:” and “main” as the only component.

1.3.3.2 PPALine() Methods

load_from_ppa()

PPALine.load_from_ppa(ppa_line=None) Loads source information from launchpad about the ppa described by the *ppa_line* argument, or from the PPALine.ppa_line attribute if blank.

- *ppa_line* - The PPA to load from Launchpad.

ppa_line

PPAs are formatted as `ppa:onwer-name/repository-name`. This argument if present, must be a string matching this format.

save_to_disk()

PPALine.save_to_disk() Fetches the signing key for the repository from the Ubuntu keyserver, then adds it to the system. It then loads the standard *Source object save_to_disk()* method to save the repository information to the system sources.

_ppa-module-functions:

1.3.4 repolib.ppa Module functions

The module containing the *PPALine()* class also contains two public helper functions that can be helpful in manual processing of the data for a *PPALine()*.

repolib.ppa.get_info_from_lp(owner_name, ppa) Requests PPA information from Launchpad and returns a Dict containing the JSON response.

- `owner_name` - The user or team name which hosts the PPA on launchpad.
- `ppa` - The name of the ppa for which to fetch information.

repolib.ppa.add_key(fingerprint) Downloads the key with `fingerprint` from keyserver.ubuntu.com and adds it to the system configuration.

- `fingerprint` - The fingerprint of the key to download and fetch. This value is present in the JSON data returned by get-info-from-lp function.

1.3.5 SystemSource()

class repolib.SystemSource() A standard *Source object* with the filename pre-set to the path for system software sources. It otherwise operates identically to the standard *Source object* class.

1.4 RepoLib Enums

RepoLib uses a variety of Enums to help ensure that data values are consistent when they need to be set to specific string values for compatibility.

1.4.1 AptSourceType

repolib.AptSourceType() Used to encode the type of packages to fetch from the archive; either binary or source code. The values for this enum are below:

- `BINARY` - Binary package source type (value: `"deb"`).
- `SOURCE` - Source code package type (value : `"deb-src"`).

1.5 Example

The following code is a Python prgram that creates an `example.sources` file in */etc/apt/sources.list.d/* with some sample data, and then modifies the suites used by the source and prints it to the console, before finally saving the new, modified source to disk:

```
import repolib
source = repolib.Source(
    filename='example.sources',
    name='Example Source', enabled=True, types=['deb', 'deb-src'],
    suites=['disco'], components=['main'],
    options={'Architectures': ['amd64', 'armel']}
)

source.suites.append('cosmic')
source.uris.append('http://example.com/ubuntu')
```

(continues on next page)

(continued from previous page)

```
print(source.make_source_string())

source.save_to_disk()
```

When run with the appropriate arguments, it prints the contents of the source to console and then saves a new `example.sources` file in `/etc/apt/sources.list.d`:

```
$
Name: Example Source
Enabled: yes
Types: deb deb-src
URIs: http://example.com/ubuntu
Suites: disco cosmic
Components: main
Architectures: amd64 armel
$ ls -la /etc/apt/sources.list.d/example.sources
-rw-r--r-- 1 root root 159 May  1 15:21 /etc/apt/sources.list.d/example.sources
```

The following will walk you through this example.

1.5.1 Creating a Source Object

The first step in using *RepoLib* is creating a *Source object*:

```
source = repolib.Source(
    filename='example.sources',
    name='Example Source', enabled=True, types=['deb', 'deb-src'],
    suites=['disco'], components=['main'],
    options={'Architectures': ['amd64', 'armel']}
)
```

The :ref:'source-object' will hold all of the information about the source we're working with.

1.5.2 Adding and Manipulating Data

The *Source object* contains attributes which describe the various parts of the source that are required to fetch and install software. Generally, these attributes are lists of strings which describe the different parts of the source. These attributes can be set or retrieved like any other attributes:

```
source.suites.append('cosmic')
source.uris.append('http://example.com/ubuntu')
```

This will add a `cosmic` suite to our source (which already has a `disco` suite added), and add a URI from which to fetch software (which wasn't previously set during object instantiation.

1.5.3 Saving Data to Disk

Once a source has the correct data (either after modification or creation), we need to save it to disk in order for Apt to be made aware of it:

```
source.save_to_disk()
```

When called, this writes the data contained within the *Source object* to disk. This does not destroy the object, so that it may be further manipulated by the program.

Note: While *Source object* data can be manipulated after using the *save_to_disk()* method, any subsequent changes will not be automatically written to the disk as well. You need to call *save_to_disk()* again in order to save further changes.

CHAPTER 2

Explanation of the DEB822 Source Format

The sources described in `/etc/apt/sources.list.d/` on a Debian-based OS are designed to support any number of different active and inactive sources, as well as a large variety of source media and transport protocols. The files describe one or more sources each and contain multiline stanzas defining one or more sources per stanza, with the most preferred source listed first. Information about available packages and versions from each source is fetched using the `apt update` command, or with an equivalent command from a different frontend.

2.1 sources.list.d

APT source information is stored locally within the `/etc/apt/sources.list.d` directory. In this directory are one or more files describing one or more sources each. For *Deb822-style Format* sources, each file needs to have the .sources extension. The filenames may only contain letters, digits, underscore, hyphen, and period characters. Files with other characters in their filenames will cause APT to rint a notice that it has ignore that file (unless the file matches a pattern in the `Dir::Ignore-Files-Silently` configuration list, which will force APT to silently ignore the file.

2.2 One-Line-Style Format

In order to understand some of the decisions behind using the *Deb822-style Format* sources, it is helpful to understand the older *One-Line-Style Format*.

One-Line-Style Format sources occupy one line in a file ending in `.list`. The line begins with a type (i.e. `deb` or `deb-src` followed by options and arguments for this type. Individual entries cannot be continued onto multiple lines (hence the "one-line" portion of this format's name). Empty lines in `.list` files are ignored, and a `#` character anywhere on the line signifies that the remainder of that line is a comment. Consequently an entry can be disabled by commenting out that entire line (prefixing it with a `#`). If options are provided, they are space-separated and all together are enclosed within square brackets (`[]`). Options allowing multiple values should separate each value with a comma (`,`) and each option name is separated from its values by an equals sign (=).

This is the traditional format and is supported by all current APT versions. It has the advantage of being relatively compact for single-sources and relatively easy for humans to parse.

2.2.1 Disadvantages

Problems with the *One-Line-Style Format* begin when parsing entries via machine. Traditional, optionless entries are relatively simple to parse, as each different portion of the entry is separated with a space. With options, however, this is no longer the case. The presence of options causes there to be no, 1, or multiple segments of configuration between the type declaration and the URI. Additionally, APT sources support a variety of URI schemas, with the capability for extensions to add additional schemas on certain configurations. Thus, supporting modern, optioned *One-Line-Style Format* source entries requires use of either regular expressions or multi-level parsing in order to adequately parse the entry. Further compounding this support is the fact that *One-Line-Style Format* entries can have one or more components, preventing parsing of sources backwards from the end towards the front.

Consider the following examples:

```
deb [] http://example.com.ubuntu disco main restricted multiverse universe
deb [ arch=amd64 ] http://example.com/ubuntu disco main nonfree
deb [lang=en_US] http://example.com/ubuntu disco main restricted universe multiverse
deb [ arch=amd64,armel lang=en_US,en_CA ] http://example.com/ubuntu disco main
```

Each of these entries are syntactically valid source entries, each cleanly splits into eight segments when splitting on spaces. Depending on which entry being parsed, the URI portion of the entry may be in index 2, 4, or 5 while options (where present) may be in index 1, 2, or 3. If we want to work backwards, then the URI is in either index -3, -4, or -6. The only segments guaranteed to be present at any given index is the type. The situation is even more complicated when considering that entries may have at a minimum 3 elements, and at a maximum 12 or more elements.

In addition to parsing difficulty, *One-Line-Style Format* entries may only specify a single suite and URI per entry, meaning that having two active mirrors for a given source requires doubling the number of entries configured. You must also create an extra set of duplicated entries for each suite you want to configure. This can make tracking down duplicated entries difficult for users and leads to longer-than-necessary configuration.

2.3 Deb822-style Format

This format addresses the file-length, duplication, and machine-parsability issues present in the *One-Line-Style Format*. Each source is configured in a single stanza of configuration, with lines explicitly describing their function for the source. They also allow for lists of values for most options, meaning that mirrors or multiple suites can be defined within a single source. A # character at the beginning of a line marks the entire line as a comment. Entries can again be disabled by commenting out each line within the stanza; however, as a convenience this format also brings an `Enabled:` field which, when set to `no` disables the entry as well. Removing this field or setting it to `yes` re-enables it. Options have the same format as every other field, thus a stanza may be parsed by checking the beginning of each line for a fixed substring, and if the line doesn't match a known substring, it can be assumed the line is an option and the line can be ignored. Unknown options are ignored by all versions of APT. This has the unintentional side effect of adding extensibility to the source; by selecting a carefully namespaced field name, third-party applications and libraries can add their own fields to sources without causing breakage by APT. This can include comments, version information, and (as is the case with *RepoLib*, pretty, human-readable names.

From the `sources.list(5)` manual page:

> This is a new format supported by apt itself since version 1.1. Previous versions ignore such files with a notice message as described earlier. It is intended to make this format gradually the default format, deprecating the previously described one-line-style format, as it is easier to create, extend and modify for humans and machines alike especially if a lot of sources and/or options are involved.

CHAPTER 3

DEB822 Source Format Specifications

Following is a description of each field in the deb822 source format.

3.1 Enabled:

Enabled: (value: "yes" or "no", required: No, default: "yes") Tells APT whether the source is enabled or not. Disabled sources are not queried for package lists, effectively removing them from the system sources while still allowing reference or re-enabling at any time.

3.2 Types:

Types: (value: "deb" or "deb-src", required: Yes) Defines which types of packages to look for from a given source; either binary: `deb` or source code: `deb-src`. The `deb` type references a typical two-level Debian archive providing packages containing pre-compiled binary data intended for execution on user machines. The `deb-src` type references a Debian distribution's source code in the same form as the `deb` type. A `deb-src` line is required to fetch source pacakge indices.

3.3 URIs:

URIs: (value: string(s), required: Yes) The URI must specify the base of the Debian distribution archive, from which APT finds the information it needs. There must be a URI component present in order for the source to be valid; multipls URIs can be configured simultaneously by adding a space-separated list of URIs.

A list of the current built-in URI Schemas supported by APT is available at the Debian sources.list manpage.

3.4 Suites:

Suites: (value: strings(s), required: Yes) The Suite can specify an exact path in relation to the URI(s) provided, in which case the *Components:* **must** be omitted and suite **must** end with a slash (`/`). Alternatively, it may take the form of a distribution version (e.g. a version codename like `disco` or `artful`). If the suite does not specify a path, at least one deb822-field-component **must** be present.

3.5 Components:

Components: (value: string(s), required: see *Suites:*) Components specify different sections of one distribution version present in a Suite. If *Suites:* specifies an exact path, then no Components may be specified. Otherwise, a component **must** be present.

3.6 Options

Sources may specify a number of options. These options and their values will generally narrow a set of software to be available from the source or in some other way control what software is downloaded from it. An exhaustive list of options can be found at the Debian sources.list manpage.

3.7 RepoLib-Specific Deb822 Fields

RepoLib presently defines a single internal-use fields which it adds to deb822 sources that it modifies.

3.7.1 X-Repolib-Name:

X-Repolib-Name: (value: string, required: no, default: filename) This field defines a formatted name for the source which is suitable for inclusion within a graphical UI or other interface which presents source information to an end-user. As a *RepoLib* specific field, this is silently ignored by APT and other tools operating with deb822 sources and is only intended to be utilized by *RepoLib* itself.

CHAPTER 4

Examples

The following specifies a binary and source-code source fetching from the primary Ubuntu archive with multiple suites for updates as well as several components:

```
Enabled: yes
Types: deb deb-src
URIs: http://archive.ubuntu.com/ubuntu
Suites: disco disco-updates disco-security disco-backports
Components: main universe multiverse restricted
```

This is a source for fetching Google's Chrome browser, which specifies a CPU architecture option and a RepoLib Name:

```
X-Repolib-Name: Google Chrome
Enabled: yes
URIs: http://dl.google.com/linux/chrome/deb
Suites: stable
Components: main
Architectures: amd64
```

This specifies a source for downloading packages for System76's Pop!_OS:

```
X-Repolib-Name: Pop!_OS PPA
Enabled: yes
Types: deb
URIs: http://ppa.launchpad.net/system76/pop/ubuntu
Suites: disco
Components: main
```

Following is a PPA source which has been disabled:

```
X-Repolib-Name: ZNC Stable
Enabled: no
Types: deb
URIs: http://ppa.launchpad.net/teward/znc/ubuntu
```

(continues on next page)

(continued from previous page)

```
Suites: disco
Components: main
```

CHAPTER 5

Apt Manage

`apt-manage` is a command line tool for managing your local software sources using RepoLib. Run `apt-manage` standalone to get a listing of all of the software repositories currently configured:

```
$ apt-manage
Current repositories:

system
pop_OS-apps
pop_OS
google-chrome
```

`apt-manage` operates strictly on DEB822-style sources on your system; however, it can accept traditional "deb lines" and `ppa:` shortcuts as input to add a repository to your system. It also has the capability of converting legacy one-line sources into DEB822 sources.

5.1 Adding Sources

The `add` subcommand is used to add new repositories to the software sources. Given without parameters, it will prompt for input to fill in the fields required to add the repository one by one:

```
$ sudo apt-manage add
Enter the URIs of the repository: http://ppa.launchpad.net/system76/pop/ubuntu
Enter the suites/distros of the repository (e.g. "disco"): disco
Enter the componenets of the repository (e.g. "main"): main
Enter a name for this repository: System76 Pop!_OS Repository
```

The source will then be saved into the system. Additionally, you can specify a deb-line to configure into the system or a `ppa:` shortcut to add the new source directly:

```
$ sudo apt-manage add deb http://apt.pop-os.org/ubuntu disco main
$ sudo apt-manage add ppa:system76/pop
```

These correctly expand into full DEB822 sources. If an internet connection is available, `apt-manage` will additionally attempt to install the signing key for any `ppa:` shortcuts added.

Note: Even though `apt-manage` will add one-line style sources to the system, it will not save them in this format. This is simply a convenience feature and is not indicative of the added source format.

Note: Because the `add` subcommand modifies system configuration, the use of `sudo` or a root user account is required.

5.2 Managing Sources

The `repo` subcommand is used for getting information about a source, or for configuring it. With no options, `repo` will list repository information:

```
$ apt-manage repo system
Name: System Sources
Enabled: yes
Types: deb deb-src
URIs:  http://apt.pop-os.org/ubuntu http://archive.ubuntu.com/ubuntu
Suites: disco disco-security disco-updates disco-backports
Components: main universe restricted multiverse
```

You can also use the `-i/--info` flag for this purpose:

```
$ apt-manage repo --info ppa-system76-pop
Name: Pop!_OS PPA
Enabled: yes
Types: deb
URIs: http://ppa.launchpad.net/system76/pop/ubuntu
Suites: disco
Components: main
```

Note: The following options modify the system configuration, and thus require you to use sudo or a root user.

5.2.1 Disabling/Enabling Sources

Disabling a source is useful if you want to temporarily stop receiving software or updates from the source. You can also use it have testing or unstable sources that you want to temporarily enable from time to time.

```
$ sudo apt-manage repo --disable ppa-system76-proposed
```

If you want to re-enable a disabled source, you can use `--enable`.

```
$ sudo apt-manage repo --enable ppa-system76-proposed
```

5.2.2 Removing Sources

You can remove sources that you want to stop using permanently.

```
$ sudo apt-manage repo --remove ppa-system76-proposed
```

5.3 Managing Source Code

To enable download source-code packages from a source, use the `source` subcommand:

```
$ sudo apt-manage source --enable ppa-system76-pop
```

To disable source-code packages:

```
$ sudo apt-manage source --disable ppa-system76-pop
```

5.4 Converting Legacy Sources

RepoLib and Apt-Manage is intended to operate strictly with DEB822-style sources. Some software may automatically add or modify existing one-line sources within your system configuration. Because of this, Apt-Manage includes a conversion utility to automatically generate DEB822 sources from any active one-line sources in your system configuration.

```
$ sudo apt-manage convert
```

This will create back-ups of any third-party source files present on your system and then convert to the new format. For extra insurance against loss of data, Apt-Manage will not delete any files, but will disable sources in any `.list` files it converts. If you need to revert to the one-line files, you can rename the relevant `.save` file to end in `.list` instead, then modify to file to uncomment any required sources. Be sure to disable the source first, or you may get warnings about sources configured in multiple locations.

> **Warning:** While every attempt has been made to ensure the process will not result in data loss, converting your sources automatically may cause problems. Back up your data and sources lists before attempting automatic conversion, or convert your sources manually.

Apt-Manage will offer to convert both third-party sources in `/etc/apt/sources.list.d` as well as system sources configured in `/etc/apt/sources.list`. It will also offer a preview of the converted source which you can use to verify that the source is correctly converted. If anything looks wrong with that source, deny the changes and then manually convert the source instead.

CHAPTER 6

Installation

There are a variety of ways to install RepoLib

6.1 From System Package Manager

If your operating system packages repolib, you can install it by running:

```
sudo apt install python3-repolib
```

6.1.1 Uninstall

To uninstall, simply do:

```
sudo apt remove python3-repolib
```

6.2 From PyPI

Repolib is available on PyPI. You can install it for your current user with:

```
pip3 install repolib
```

Alternatively, you can install it system-wide using:

```
sudo pip3 install repolib
```

6.2.1 Uninstall

To uninstall, simply do:

```
sudo pip3 uninstall repolib
```

6.3 From Git

First, clone the git repository onto your local system:

```
git clone https://github.com/isantop/repolib
cd repolib
```

6.4 Debian

On debian based distributions, you can build a .deb package locally and install it onto your system. You will need the following build-dependencies:

- debhelper (>=11)
- dh-python
- python3-all
- python3-setuptools

You can use this command to install these all in one go:

```
sudo apt install debhelper dh-python python3-all python3-setuptools
```

Then build and install the package:

```
debuild -us -uc
cd ..
sudo dpkg -i python3-repolib_*.deb
```

6.4.1 Uninstall

To uninstall, simply do:

```
sudo apt remove python3-repolib
```

6.5 setuptools setup.py

You can build and install the package using python3-setuptools. First, install the dependencies:

```
sudo apt install python3-all python3-setuptools
```

Then build and install the package:

```
sudo python3 ./setup.py install
```

6.5.1 Uninstall

You can uninstall RepoLib by removing the following files/directories:

- /usr/local/lib/python3.7/dist-packages/repolib/
- /usr/local/lib/python3.7/dist-packages/repolib-*.egg-info
- /usr/local/bin/apt-manage

This command will remove all of these for you:

```
sudo rm -r /usr/local/lib/python3.7/dist-packages/repolib* /usr/local/bin/apt-manage
```