

---

# **keepitfresh Documentation**

***Release 1.0.2***

**Daniel Nunes**

**Aug 19, 2019**



# CONTENTS

<b>1</b>	<b>Usage</b>	<b>3</b>
<b>2</b>	<b>Reference</b>	<b>5</b>
	<b>Index</b>	<b>7</b>



To install *keepitfresh*, use pip:

```
pip install keepitfresh
```

Simple as that! You now have *keepitfresh* available in your environment.



## USAGE

You can find a more thorough description of each argument below, this section illustrates an example usage with some pseudo-code:

```
>>> base_url = 'http://www.example.com/'
>>> regex = r'(\d+\.\d+\.\d+)\.(?:tar\.gz|zip|rar|7z) '
>>> current_version = '0.0.1'
>>> overwrite_item = 'path/to/application'
>>> entry_point = 'example.exe'
>>> # check if it can be updated
>>> is_fresh(base_url, regex, current_version):
False

>>> # current version is not fresh, let's update
>>> payload = {'base_url': base_url 'regex': regex, 'current_version': current_
↳ version, 'overwrite_item': overwrite_item, 'entry_point': entry_point}
>>> freshen_up(**payload) # process will restart automatically
```

Usually you should only call `is_fresh()` if you're not updating. Otherwise do this:

```
>>> try:
...     freshen_up(**payload)
... except RuntimeError:
...     # no new version
...
```

For some further examples on base url and regex combos take a look at this [page](#), originally meant for *uscan* but also usable for this package.





## REFERENCE

`keepitfresh.freshen_up(**kwargs)`

Finds, downloads, unpacks, overwrites and restarts your application. Essentially an all-in-one for your convenience.

This function requires 5 arguments to be passed with an additional 2 optional.

The required arguments are as follows:

- **base\_url** - The url that contains the links to download the package in the form `<a href="...">`.
- **regex** - The regular expression that matches the file name. Must contain at least one capturing group representing the version string and this must be the first group.
- **current\_version** - The current version of the application as a string.
- **overwrite\_item** - The file/folder where your application is and that is going to be overwritten.
- **entry\_point** - The relative path from **overwrite\_item** to the executable that restarts the application.

The optional arguments are as follows:

- **versioncmp** - A function to override the default version comparison method, that takes 2 positional arguments, two version strings, and returns `True` whenever the second version string is newer than the first version string.
- **unpack** - A function to override the default unpacking method that takes two arguments, the archive path and the output folder.

If **versioncmp** is not provided, the standard comparison method from the `packaging` package is used. If **unpack** is not provided, unpacking is handled by `patool`.

`keepitfresh.is_fresh(base_url, regex, current_version, versioncmp=None)`

Checks whether your application is fresh (if there is a more recent version). Returns `False` if there is a newer version, `True` otherwise.

For what each argument means, please refer to `freshen_up()`.

`keepitfresh.get_file_urls(base_url, regex)`

Inspired by `uscan`, the debian packaging utility.

Looks through all `<a href="*">` references to files in the given base url and extracts them into a dictionary of (file\_url, file\_version) value-pairs.

The **regex** argument is a regular expression that matches the file name. It **MUST** have the file's version in a capturing group and this **MUST** be the first group (`\1` backreference).

As an example, consider a project named *b* by *a* which deploys to Github Releases with filenames such as *b-1.0.0.zip*. The function call would look like:

```
>>> base_url = "https://github.com/a/b/releases"
>>> regex = r"b-(\d+\.\d+\.\d+)\.zip"
>>> result = get_file_urls(base_url, regex)
>>> result
{"https://github.com/a/b/releases/download/1.0.0/b-1.0.0.zip": "1.0.0"}
```

`keepitfresh.get_update_version(file_dict, current_version, vcmp=None)`

Look through a dictionary that maps file urls to version strings, much like the one returned by `get_file_urls()`, and get the latest version and corresponding file url. If no version newer than **current\_version** is found, returns an empty tuple.

**current\_version** should be a string in the same pattern as used in `get_file_urls()`.

To get the latest version, a comparison function is used. The default uses the comparison from the `packaging` package. To override this, pass a function in **vcmp** that accepts two version strings and returns `True` whenever the second version string is newer than the first version string.

`keepitfresh.dl_unpack(url, outdir, unpack=None)`

Downloads the archive in **url** and unpacks it to **outdir**.

Unpacking is handled by `patool`. If you need to override this, you can a function in **unpack** that accepts the archive path as the first argument and the output folder as the second argument.

`keepitfresh.overwrite_restart(initem, owitem, entry_point)`

Overwrites the current application file/folder and restarts the process with the updated application.

Inspired by PyUpdater, uses a separate process for Unix and Windows (Windows does not allow file deletion while it's still being used so we have to work around that).

**initem** can be either a file or a folder and is the path to the updated application. **owitem** can be either a file or a folder and is the path to the old application.

**entry\_point** is the relative path from the parent folder of **owitem** to the executable to restart with.

## INDEX

### D

`dl_unpack()` (*in module keepitfresh*), 6

### F

`freshen_up()` (*in module keepitfresh*), 5

### G

`get_file_urls()` (*in module keepitfresh*), 5

`get_update_version()` (*in module keepitfresh*), 6

### I

`is_fresh()` (*in module keepitfresh*), 5

### O

`overwrite_restart()` (*in module keepitfresh*), 6