check4updates Documentation

Release latest

Apr 03, 2021

USAGE

check4updates

check4updates is a Python library for developers that prompts your users to upgrade when a new version of your package is released.

check4updates

CHAPTER

ONE

WHAT IS CHECK4UPDATES?

check4updates is a Python package designed for developers to provide a reliable, simple, and unobtrusive way to check whether their users have the most recent release of their package installed and prompt users when an update is available. It is recommended that *check4updates* be placed in your package's `__init__.py` file so that it is called every time the package is used. To minimise runtime, *check4updates* only checks online if certain conditions are met, such as sufficient time since the last check. This ensures that incorporating *check4updates* into your Python package will have a negligible (<0.01sec) effect on the time it takes users to import something from your package (which is when `__init__.py` is called) and no impact on any of the functions within your package. When *check4updates* does check online, it searches PyPI for the most recent release (based on version number) and compares that value with the version installed on the user's system. If certain conditions are met, the user is then prompted to install the updated version which must be done by the user in a separate command so as not to interrupt the currently executing script. Users can choose to upgrade now, skip this version, be reminded later, or never be asked again and *check4updates* will remember this choice and act accordingly.

Please see how does *check4updates* work for a more detailed description of its functionality.

CHAPTER

TWO

HOW DOES IT WORK?

check4updates only contains two classes. One of these is *check_and_prompt()* and the other is *upgrade()*. These are discussed below:

2.1 check4updates.check_and_prompt()

Remember to include *check4updates* in your package's requirements. To use *check4updates*, place the following lines in your package's `__init__.py` file:

```
from check4updates import check_and_prompt
check_and_prompt("your_package")
```

Each time your users import something from your package, the `check_and_prompt` command is executed. This command does the following:

1. Check if there is a record of a previous action by searching the package's package folder (where `__init__.py` is located) for the file "check4updates.txt".
2. If "check4updates.txt" is not found, go to (3) else go to (6).
3. *check4updates* will use the requests library to search PyPI for the latest version and check which version the user currently has installed. If there is an error in getting the PyPI version go to (8) else if the PyPI version is obtained successfully, *check4updates* will compare the versions to determine whether an update is available. If an update is available go to (4) else go to (5).
4. When an update is available, prompt the user with the following:

```
Version 0.2.1 of example_package is available on PyPI
You currently have version 0.1.9 of example_package installed
Please choose an option:
1. I want to upgrade
2. Remind me tomorrow
3. Skip this version
4. Never ask me again
Your choice:
```

The user's choices will trigger the following:

- Update now ⇒ Instructions will be printed to the console on how to update. See below for text outputs. END
- Remind me later ⇒ *check4updates* will store the current timestamp and the action to "remind" in the "check4updates.txt" file and place this in the package folder. This is the file that is checked in step (1). END
- Skip this version ⇒ *check4updates* will store the current time and the action to "skip" in the "check4updates.txt" file and place this in the package folder. This is the file that is checked in step (1). END

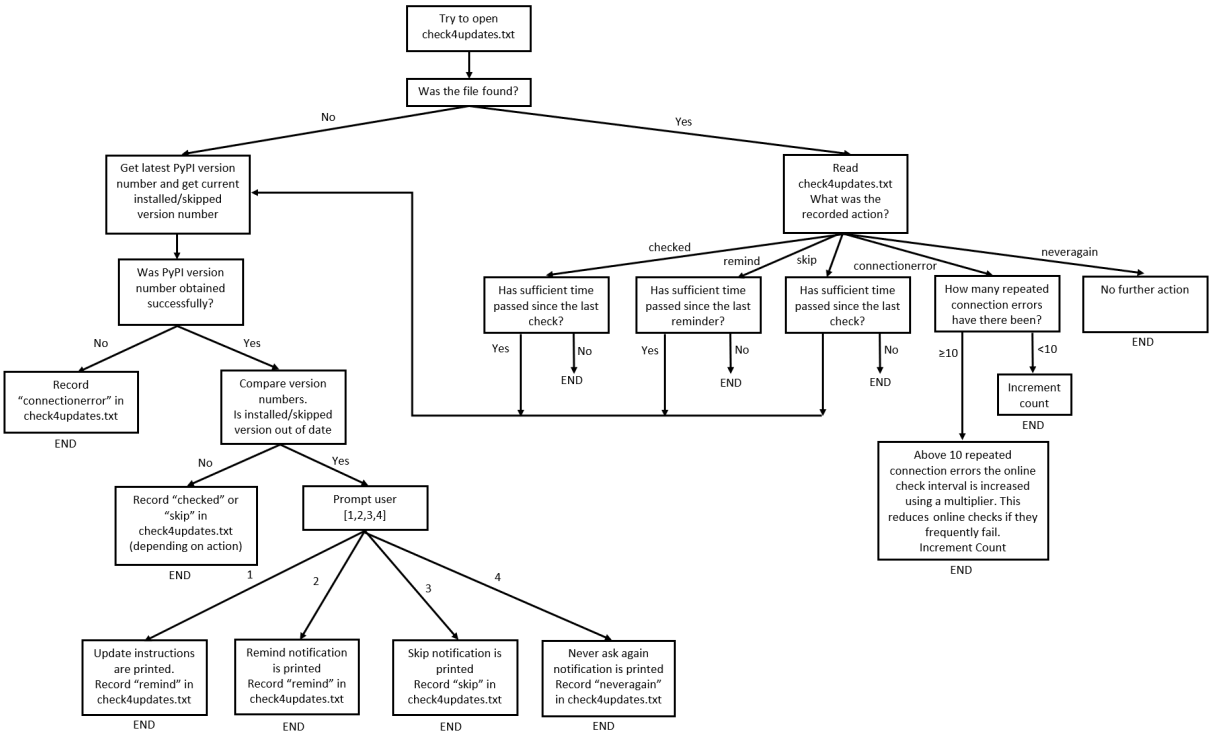
- Never ask me again ⇒ *check4updates* will store "neveragain" in the "check4updates.txt" file. This will be remembered even if the user updates the package manually. END

5. If no update is available, *check4updates* will store the current time and the action "checked" in the "check4updates.txt" file and place this in the package folder. This ensures that the time of the last check is recorded and will be found in step (1) on the next execution. END
6. The file "check4updates.txt" contains 4 pieces of information on a single line separated by spaces. These are "action timestamp PyPI_version count". When *check4updates* is found to contain these actions, the following steps will be taken:

- checked - determine how much time has passed between the timestamp of the last check and now. If sufficient time has passed (as determined by the "online_check_interval" argument) then another check will be performed. Go to (3) else END.
- remind - determine how much time has passed between the timestamp of the last prompt and now. If sufficient time has passed (as determined by the "remind_delay" argument) then another check will be performed (in case a new version was released since the last prompt). Go to (3) else END.
- skip - determine how much time has passed between the timestamp of the last check and now. If sufficient time has passed (as determined by the "online_check_interval" argument) then another check will be performed. This ensures that check4updates continues to periodically check PyPI for a new version when the user has selected to skip the current version. *check4updates* will use the requests library to search PyPI for the latest version and check whether the latest version from PyPI exceeds the PyPI_version stored in "check4updates.txt". If there is an error in getting the PyPI version go to (8) else if a new version (beyond the skipped version) is available then go to (4) else go to (7).
- neveragain - No further action will be taken as the user has previously decided never to be asked again. END
- connectionerror - This means *check4updates* has previously tried to seach online in step (3) and has encountered a problem with the connection (likely due to not being connected to the internet). In this case the action was recorded as connectionerror. Another check will be performed if sufficient time has passed (as determined by the "check_interval" argument). Note that the count is used to increase the value of the "online_check_interval" argument if there have been repeated failed connection attempts. This is intended to reduce the number of attempts to check PyPI when failure seems likely. END

7. As the user has selected to skip the current PyPI version, update the timestamp so that the next time (6) is run it will only check online if sufficient time has passed. END
8. If there is an error in getting the PyPI version then "check4updates.txt" will have connectionerror recorded in the action, the current timestamp, 0.0.0 as the PyPI version, and 1 added to the current count. END

The flowchart below shows the above description in an abbreviated form.



2.2 check4updates.upgrade()

`check4updates.upgrade()` provides a simple way of updating a package from within a new Python script. The option to use *upgrade()* is provided in step (4) if the user selects "update now". To run *upgrade()* from within a script, type the following:

```
from check4updates import upgrade
upgrade("your_package")
```

When this command is run it calls pip as a subprocess and passes the "your_package" argument. This is the python script equivalent to typing in your command prompt or terminal.

```
pip install --upgrade your_package
```

The output from pip that you would normally get in your command prompt or terminal is printed to your IDE's console.

2.3 Error handling

check4updates is designed to never impact the runtime of the parent script. *check4updates* achieves this by handling errors silently, ensuring the user will never receive an error (such as no internet connection when trying to check online). This gives developers the confidence that using *check4updates* in their packages will never result in their package being crashed by *check4updates*, thereby avoiding negative user experiences.

The one downside to this is error reporting (to the user) from *check4updates* is non-existent. This means that if *check4updates* does run into trouble then you will probably never know. Your users would only know if they were diligent enough to check whether an update was available and to note that *check4updates* was part of the `__init__.py` file but they were not receiving a prompt to update. This could impact some of your users (based on their unique

system configuration) but the worst case scenario (of failing to notify users of an update) is equivalent to not using *check4updates* at all.

In accordance with the MIT license, the author of *check4updates* provides no guarantees or assurances that the use of this software will not cause errors. All effort has been made to ensure the software is free of errors, however, the software is provided "as is", without warranty of any kind, express or implied.

2.4 Text outputs

When the user is prompted for their choice, they receive the following text:

```
Version 0.2.1 of example_package is available on PyPI
You currently have version 0.1.9 of example_package installed
Please choose an option:
1. I want to upgrade
2. Remind me tomorrow
3. Skip this version
4. Never ask me again
Your choice:
```

The following text outputs will be printed to the console when the user selects 1, 2, 3, or 4 from the prompt:

- choice = 1

```
To upgrade example_package you can do one of the following:
Open your command prompt / terminal and type: pip install --upgrade␣
↪example_package
or
From within your Python IDE in a new Python script type:
from check4updates import upgrade
upgrade('example_package')
Then run the script and example_package will be upgraded to the most␣
↪recent version.
```

- choice = 2

```
You will be reminded again tomorrow or the next time you use example_
↪package
To upgrade to version 0.2.1 manually, please use: pip install --upgrade␣
↪example_package
```

- choice = 3

```
Version 0.2.1 of example_package will be skipped
You will be prompted again when the next version of example_package is␣
↪released
To upgrade to version 0.2.1 manually, please use: pip install --upgrade␣
↪example_package
```

- choice = 4

```
You will never again be prompted to upgrade example_package, even if you␣
↪upgrade manually.
To upgrade to version 0.2.1 manually, please use: pip install --upgrade␣
↪example_package
```

check4updates

CHAPTER

THREE

WHY CAN'T IT AUTO UPDATE?

You may think "why can't *check4updates* perform the update automatically so my users don't have to be prompted?". Updates must be done manually by the user for 3 reasons:

- 1. It is not possible to update a package that is currently in use. If a package calls an update script during its execution, then the update script calls pip and pip will attempt to delete the old installation as part of the update. This causes an error (at least it does in Windows) because the file to be deleted is currently in use. It is possible to update a package using a script but that script can not be called from within the package to be updated.
- 2. It would interrupt the currently executing script. Any self updating script called mid-way through some other parent script will be faced with the difficult job of ensuring the currently executing script's state and variables are not destroyed by the update process. This may be achieveable by ensuring any update is only performed when the currently running script is finished executing, though Python usually doesn't work this way without some complex threading of subprocesses.
- 3. It would be a security vulnerability for the Python Software Foundation to allow the creation of scripts that can self update in the background without seeking the user's permission. If this was possible it could enable a malicious developer to introduce code on the user's system without their knowledge.

CHAPTER

FOUR

SIMILAR LIBRARIES

There are many other Python libraries on PyPI which are designed to perform some form of package updating. These include:

- update-checker
- pip-update
- req-update
- update-check
- esky
- upgrade-requirements
- pip-upgrade
- updater

Most of these libraries are intended for bulk updating all your installed packages. Some are intended for updating a very specific package. None of these similar libraries overlap with the function and design principles of *check4updates*. These design principles are:

- periodically check for updates without user action and without slowing down the parent script
- only prompt the user when necessary
- give the user the option to update now, be reminded later, skip this version, or never be asked again
- handle all errors silently to prevent the parent script from being impacted

CHAPTER

FIVE

CHANGELOG

5.1 Version: 0.0.2 – released 03 Apr 2021

- Testing using GitHub Actions to ensure compatability with different Python versions and different operating systems
- Added a mock_user_input argument to enable easy testing without the need to prompt the user
- All printing is in red
- Improved regex to deal with detection of certain version patterns
- Improved padding of version strings to deal with comparison of certain version patterns
- Added an option for the user to never be prompted again

5.2 Version: 0.0.1 – released 24 Mar 2021

- Initial commit of project files for Alpha testing.

CHAPTER

SIX

CONTRIBUTING

Suggestions for improvements and pull requests are most welcome, though I prefer to discuss your suggestion before you go to the trouble of creating a pull request. If you would like to discuss a change to the code, please email me (alpha.reliability@gmail.com).

If you find a bug or encounter a problem, please raise an issue on GitHub.