Theory: Pip

9 minutes 0 / 5 problems solved

Skip this topic Start practicing

One astonishing fact about Python is that it has a huge and diverse community of contributors. Essentially, that means that there are plenty of solutions to a significantly vast range of problems in open access. This fact comes in handy, especially when you are working on your own projects. It's highly likely that you'll be able to find a proper task-specific package and use it effectively to meet your needs. Now we are going to learn about standard tools for package management in Python.

§1. What is pip

By this time you've probably familiarized yourself with the Python standard library. It contains a lot of useful built-in modules and should be preinstalled with your Python distribution. In fact, one more thing that is preinstalled (starting with **Python 3.4**) is the standard package manager called **pip** (the acronym is commonly expanded as `"Pip Installs Packages"`).

Pip is designed both to extend the functionality of the standard library by installing the additional packages on your computer and to help you share your own projects and thereby contribute to the development of Python.

Now let's make sure that you have **pip** installed. All you need to do is open a command prompt/terminal and run this line:

```
pip --version
```

The output should report your current pip version. For example, the latest version is:

```
pip 20.0.2
```

In case it's not installed (or you want to upgrade it), please follow these installation instructions specifically for your operating system.

If your terminal cannot find the `pip` command, try to use `pip3` instead.

§2. Pip capabilities

Since **pip** is the recommended installer for Python, the most obvious and crucial command to begin with is `install`. Have a look at the following line:

```
pip install some_package
```

The installation is really that simple. However, if you are interested in a certain version of the package, you need to specify it after the package name like this:

```
pip install some_package==1.1.2
```

Or, at least, define a minimal suitable version:

```
pip install "some_package>=1.1.2"
```

Note that the last expression should be enclosed within double quotes for the comparison operator to be interpreted without any problem.

Another useful thing is the `show` command. It shows information about installed packages, for instance, their version, author, license, location or requirements. Here is a general example:

```
pip show some_package
```

Current topic:

Pip Stage 1

Topic depends on:

- ✓ Packages Stage 1
- ✗ Parameters and options Stage 1

Topic is required for:

- Requests: retrieving data
- Requests: manipulating data
- XML in Python
- Built-in exceptions
- Introduction to pandas
- Introduction to sklearn
- Introduction to Django Stage 1
- Intro to NumPy
- Overview of NLTK

Table of contents:

Also, the `list` command might be of use. It lists all the packages you've installed on your computer in alphabetical order:

```
pip list
```

If you print the `list` command with the option `--outdated`, or just `-o`, you'll get the list of outdated packages coupled with both the current and latest versions available.

```
pip list --outdated
```

or with a bit shorter variant:

```
pip list -o
```

After executing one of the mentioned lines, you will see a similar output:

```
first_package (Current: 2.1.1 Latest: 3.0.1)
second_package (Current: 4.2.1 Latest: 4.2.2)
```

Having discovered outdated packages, you might want to update them to the newest available version:

```
pip install --upgrade some_package
```

To remove a package from your computer run the `uninstall` command:

```
pip uninstall some_package
```

When developing your project, it may be advantageous to keep a list of packages to be installed, i.e. dependencies, in a special file (see Requirements File Format). It is convenient because you can install the packages directly from it:

```
pip install -r requirements.txt
```

Of course, you are not supposed to write this file yourself listing all the necessary packages. It will be enough to run the code below in order to obtain it:

```
pip freeze > requirements.txt
```

Let's examine the line above in detail. `freeze` is a command used to get all installed packages in the format of requirements. So all the packages you had installed before the execution of the command and presumably had used in some projects would be listed in the file named **`"requirements.txt"`**. Furthermore, their exact versions would be specified (see Requirement Specifiers).

Consider an example output of the `freeze` command:

```
beautifulsoup4==4.7.1
nltk==3.4.1
numpy==1.16.3
scikit-learn==0.21.1
scipy==1.3.0
```

What's important is that `freeze` actually lists **all** the installed libraries, which is rarely necessary and might be considered a bad practice. For this reason, we recommend that you take a more conscious approach and revise the obtained ***requirements*** file by yourself.

§3. Recap

Overall, we've learned the basics for package installation through **pip**:

- how to install packages (either a specific version or non-specific one),
- how to create a ***requirements*** file and use it for installation,

- how to obtain information about installed packages,
- and, finally, how to uninstall packages.

For further details, try consulting the documentation or running the command `help`.

Now let's get to practice so that you can use all this information in the future!

Report a typo

215 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

This content was created over 1 year ago and updated about 21 hours ago. Share your feedback below in comments to help us improve it!

Comments (12) Hints (4) Useful links (0) Show discussion