manati

Matthias Baer

Apr 29, 2021

CONTENTS:

manati is a command line interface (CLI) for managing Python projects.

CHAPTER

ONE

GETTING STARTED WITH *MANATI*

manati is a command line interface (CLI) for managing Python projects.

Create new Python projects with ready-to-go recommended project structure.

Add important files to existing projects like `setup.py`, `.gitignore`, Sphinx documentation, choose a license and more.

Run test suites, analyze test coverage and **deploy** to PyPi.

Even *manati* is managed using *manati*... so meta.

1.1 Installation

```
pip install --upgrade manati
```

1.2 Usage

1.2.1 Creating a new project

```
manati create -n myproject
```

creates a complete Python project structure inside the current working directory:

```
myproject
├── docs
│   ├── Makefile
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── requirements.txt
├── myproject
│   ├── __init__.py
│   └── main.py
├── LICENSE
├── README.md
├── setup.py
├── .gitignore
└── tests
    └── test_main.py
```

including sample source, tests, documentation, `setup.py`, local `git` repository and a suitable `.gitignore` file.

After creation, the project is already installed in development (editable) mode, so you can start coding right away.

1.2.2 Adding stuff to an existing project

Sometimes you have an existing project, but initially you did not choose a license, or your `.gitignore` is missing. You can add those special files with the `manati add` command.

Add a license

```
manati add license
```

where you have the choice between standard license texts like MIT, GPLv3, Apache, ...

Add a `.gitignore` file

```
manati add gitignore
```

The created `.gitignore` contains all usual patterns that should typically be ignored by git in Python projects.

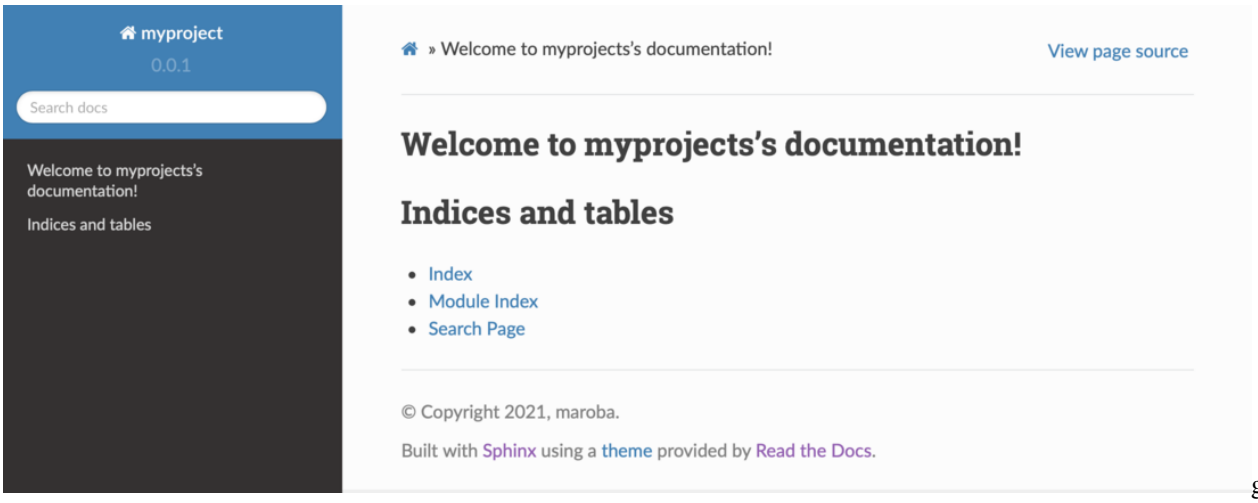
Add a `setup.py` file

```
manati add setup.py
```

Add a project documentation folder

```
manati add docs
```

makes a `./docs` folder and sets up a *Sphinx*-based documentation in `Read-The-Docs`-style:



generated docs

Call `manati add --help` for more information.

Add github actions

Add a standard github action which automatically runs build and tests on the github CI/CD systems whenever you push a commit:

```
manati add github-action
```

1.2.3 Run stuff

Run tests

```
manati run tests
```

Analyze test coverage

```
manati run coverage
```

Re-Build docs and show it browser

```
manati run docs
```

Run PEP8 style analyzer

```
manati run flake8
```

1.2.4 Deploy your project

To PyPi

```
manati deploy pypi
```

After that anyone in the world can install your package using *pip*.

As a prerequisite for deployment, you need an account at *PyPi*.

To Github, Gitlab, etc.

Create an empty repository at the platform of your choice, like github, and deploy your local project repository there, e.g.:

```
manati deploy repo
```

After that your local repo is in sync the remote one.

1.3 Note for Windows users

Depending on your environment settings, you may have to use *manati* by prepending `python -m` or `py -m` like in

```
python -m manati create
```

CHAPTER

TWO

CREATING A NEW PROJECT

manati can set up a default and ready-to-go structure for Python projects that contains probably everything you need (if not, submit an issue at github :-)).

Suppose you want to create a project named *myproject*. Go to the directory where you want the new project to be created and type

```
manati create
```

You will be prompted for a few questions for setting up the project (defaults in square brackets can simply be accepted by pressing ENTER):

```
Project name: myproject
Author [mbaer]: maroba
(Short) description []: My fancy new project
License (MIT, GPLv3, Apache, None) [None]: MIT
```

After that *manati* sets up the following directory structure:

```
myproject
├── docs
│   ├── Makefile
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── requirements.txt
├── myproject
│   ├── __init__.py
│   └── main.py
├── LICENSE
├── README.md
├── setup.py
├── .gitignore
└── tests
    └── test_main.py
```

The `./docs` folder contains documentation for the project in Read-The-Docs style based on Python's quasi standard Sphinx, and a first HTML version has also been built. You can watch it by opening `./docs/_build/html/index.html` in your browser or, more easily by running

```
manati run docs
```

which will (re-)build the docs and open it up in a browser.

The `.gitignore` file that has been created contains most of the file patterns which should not be part of a *git* repository of Python projects.

A local *git* repository has also been created as you can see by typing

```
cd myprojects
git status
```

The newly created project also contains a `setup.py` in the project root directory, that is used for installation in development mode and also for later deployment to a package index like PyPi. You may want to adjust some settings in the `setup.py` file, like your email address, the project URL or maybe the intended audience classifiers. You can look up valid classifiers at PyPi.

After creation, *manati* has already installed it in development mode, so you can start coding and any changes will be automatically be taken into account without the need to re-import anything.

A sample code module `myproject/main.py` has been created along with a test module `tests/test_main.py`. You can run the test suite with your favorite testing framework, for instance with the *unittest* framework from Python's standard library:

```
python -m unittest discover tests
```

or alternatively with *manati*:

```
manati run tests
```

where you have the choice between different testing frameworks.

Creating a new project can also be done in one line by specifying the required information as options:

```
> manati create --help

Usage: manati create [OPTIONS]

  Create a standard Python project structure.

  By default, the project is also pip-installed for development in editable
  mode, and a local git repository is also created.

Options:
  -n, --name TEXT                 Name of the project, same as the main
                                  package.  [required]

  -G, --no-git                    Do not create git repository
  -I, --no-install                Do not pip-install in editable mode
  -a, --author TEXT
  -d, --description TEXT
  -l, --license [MIT|GPLv3|Apache|None]
  --help                          Show this message and exit.
```

CHAPTER

THREE

ADDING STUFF

If you already have a project, but it is missing some important aspects like proper `setup.py`, license file, `.gitignore` or other, you can add that easily with *manati*.

This is what you can add with *manati* in the current version:

```
> manati add --help

Usage: manati add [OPTIONS] COMMAND [ARGS]...

  Adds something to the current project.

Options:
  --help  Show this message and exit.

Commands:
  docs           Add a docs folder with Sphinx documentation to the current...
  github-action  Add github default action
  gitignore      Add a default .gitignore file to the current directory.
  license        Add a license to the current project.
  package        Add a package to the current directory.
  setup.py       Add a setup.py file to the current directory
```

3.1 Adding documentation

Documentation for Python projects (not docstrings) in usually stored in the `./docs` folder under the project root directory. The quasi standard for generating documentation in Python is Sphinx. If your project is missing a documentation, you can set it up with

```
manati add docs
```

and the folder and an initial documentation is set up. You can modify the documentation settings and plugins used by editing the `./docs/conf.py` file, which is the central configuration file for Sphinx.

Now the initial documentation is ready to be generated, which can be done by typing

```
manati run docs
```

from the project root directory. After the building process, a browser opens up showing the resulting HTML files, which should look very similar to the documentation you are currently reading.

3.2 Adding *.gitignore* file

It is important not to clutter your *git* repository with files that are not needed for the code itself. If your project does not yet have a suitable `.gitignore` file that catches must of the irrelevant stuff, you can add one to your project by typing

```
manati add gitignore
```

3.3 Adding *setup.py* file

When you want to install your project to your Python environment so that it can be used from anywhere in your file system, you need to have a `setup.py` file for proper installation. You can add such a file to your project by simply typing

```
manati add setup.py
```

After that, you may want to edit some of the settings in the file, like author, email, name of the package, etc.

3.4 Adding a license

Defining a license for your project is important if you want other people to use your code and control how they may use it. If your project is missing a suitable license file, you can add one by

```
manati add license
```

You have the choice between several different typical license types. If you are unsure which one to select, take a look at choosealicense.com.

3.5 Adding a package

Suppose you have a package `mypackage`. If you want to create a subpackage `foo`, and inside this one another subsubpackage `bar`, you can do this in one step with *manati*:

```
manati add package mypackage.foo.bar
```

The proper `__init__.py` files are also created of course.

3.6 Adding github-actions

When you want to host your *git* repository on github.com, you may want to use their continuous integration / continuos deployment tools, Github Actions. To use that, your repository needs a hidden directory `./.github/worksflows/` containing configuration files that define what actions to perform. With *manati* you can quickly set that up for your project by simply typing

```
manati add github-action
```

You will be asked about the name of the package and the folder with the tests, so that the tests can be run properly. You may not see the folder immediately, because it is hidden, but it is there. :-)

After you commit and push your repository to github, the action will be triggered. And it will also be triggered on every future push.

CHAPTER

FOUR

RUNNING STUFF

Depending on what helper tools you are using, you have to use different syntax, which is kind of annoying. For instance, I regularly forget the proper way how to trigger the test discovery with the *unittest* framework of the Python standard library. With *manati* you have the choice to use different tools, but don't have to remember the exact syntax for each of them.

Here is what you can currently run with *manati*:

```
  coverage  Run test coverage.
  docs      Build the documentation and show it in browser.
  flake8    Run PEP8 style enforcement.
  tests     Run tests in a test folder.
```

4.1 Run tests

To run your test suite with *manati*, type

```
manati run tests
```

You will be asked what testing framework you want to run your tests with. Currently, *manati* supports *unittest* and *pytest* as two of the most popular solutions. If you are missing your favorite framework, please submit an issue with a feature request.

```
Options:
  -t TEXT                         Directory with tests.  [required]
  -r, --runner [unittest|pytest]  Test runner  [required]
  --help                          Show this message and exit.
```

4.2 Analyze test coverage

You can analyze the test coverage with *manati* by typing

```
manati run coverage
```

You will be asked which package to analyze and in which folder the tests are located. If the defaults that *manati* is guessing are correct, just confirm them with ENTER.

```
Options:
  -s, --source TEXT               Package on which to run coverage.
```

(continues on next page)

(continued from previous page)

```
                                [required]

  -t, --tests TEXT              Directory with tests.  [required]
  -r, --runner [unittest|pytest]  Test runner  [required]
  --help                        Show this message and exit.
```

4.3 Run docs

If your project has a `./docs` folder with a proper Sphinx documentation, you can build the HTML files and show them in the browser by simply typing

```
manati run docs
```

4.4 Run style enforcement

Proper code style is important, so you should follow the style recommendations as defined in PEP8. However, the line length limitation of 79 characters is extremely annoying. That is why *manati* uses 120 character per line as the default in *flake8*. To scan for style deviations, run

```
Usage: manati run flake8 [DIRS]...

  Run PEP8 style enforcement.

  But in contrast to PEP8, by default 120 characters per line are ok.
```

CHAPTER

FIVE

DEPLOY YOUR PROJECT

When your code is ready to share with other Python developers, you should submit your repository to a platform like github.com, so that people can contribute to your project. And when you want people in the world to be *pip install* your package, you should submit your code to the package index PyPi. Both things can be done with *manati*:

```
Usage: manati deploy [OPTIONS] COMMAND [ARGS]...

  Deploy your project.

Options:
  --help  Show this message and exit.

Commands:
  pypi  Deploy project to PyPi package repository.
  repo  Deploy local git repository to github, gitlab, bitbucket, etc.
```

5.1 Deploy to PyPi

From the root directory of your project (where the `setup.py` is located), type

```
manati deploy pypi
```

In the background, *manati* will install and update any requirements needed deployment and then build your code and create folders `dist` and `build`. Then it will upload the package to PyPi, that's why you are asked to enter your PyPi username and password. Of course, this means that you have to have a PyPi account first, which you can register here.

If everything goes right, at the end *manati* prints a link to your submitted package at pypi.org. It is online now and can be installed with *pip* by anyone in the world.

So what can go wrong? In the `setup.py` file (if you don't have one yet, *add it with manati*), you need valid values for at least some of the variables, like an email address, a URL for the project (like the github repository) and of course, the name variable should be the same as the package that you want to submit. The version variable is also crucial, because you can only upload code of a given version once. So you may have to adjust it. In any case, make sure that the name of your package is not already taken by someone else (check on pypi.org).

If you have set up the project structure with *manati*, you are already done with all settings and ready to deploy.

5.2 Deploy to github, gitlab, bitbucket, etc.

When you create a project with *manati*, it also creates a local git repository on your computer. You can use it as it is, but at some point, you may want to have your repository in the internet so that other people can see it, use it and maybe contribute to it. *manati* can help you transferring your code to one of the git-based platforms.

First thing to do: go to github.com or gitlab.com or whatever and create a new EMPTY repository there. Copy the URL for your new repository. Make sure you have commited your latest code changes to the local repo and then type

```
manati deploy repo
```

manati will ask you for the repository URL and the name of the default branch (depending on the platform you use, this may be *main* or *master*). After that the code is copied to the platform and your local repository is configured to track the remote one. So as of now you can use *git push* and *git pull* to transfer changes between local and remote repositories.

CHAPTER

SIX

HOW TO CONTRIBUTE

manati is open source and everyone is welcome to contribute in its development!

6.1 Report bugs and feature requests

If you are missing a feature or have found a bug, please submit an issue. There are no formal requirements for how an issue should look like. If you have found a bug, please also give input, output and error messages, if possible. Maybe you already know how to solve it. In that case you could describe the solution, or alternatively, fix it yourself (see description below).

6.2 Contributing code or documentation

Of course, you can clone the *manati* repository to your local machine and change the code there, but then you cannot feed your changes back to the original repository. When you want your changes to be part of *manati*, you should **fork** the *manati* repository instead. This creates a copy of the repository in your own space on github.

6.2.1 Set up development environment

Clone *your fork* of *manati* to your machine, go to the project root directory of *manati* (where `setup.py` is located) and type

```
pip install -e .
```

This installs *manati* in editable / development mode. So any changes to the code will automatically be active for any local users of the code. That's all.

6.2.2 Developing code

Please be sure to write tests for all your changes. The tests are located in `./tests` and there is one test module per code module. The naming convention is `./tests/test_MODULENAME.py` if you have a module `./manati/MODULENAME.py`.

manati uses the *unittest* framework from the Python standard library. So please stick with that. You can run the test suite from the project root directory by

```
manati run tests
```

Remember, even *manati* can be managed using *manati*. :-)

We strive for a high test coverage, so please make sure that your changes do not decrease the percentage of covered lines. You can run the coverage by

```
manati run coverage
```

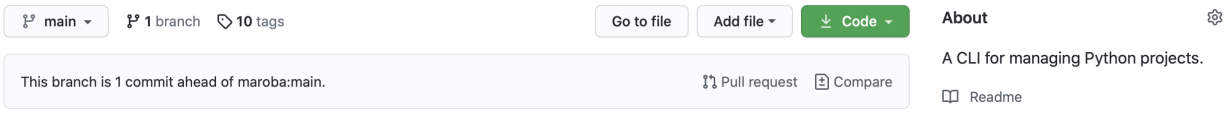
And needless to mention: Please do not commit failing tests!

When changing the code base, please follow typical PEP8 style conventions, **except for the max. line length rule**. *manati* code should have a maximum line length of 120 characters. You can check for style violations by typing

```
manati run flake8
```

from the project root directory.

Once you are ready with your changes, covered by tests and **all** the tests are running, send a pull request from your fork:



pull request

6.2.3 Writing documentation

The documentation is in the `./docs` folder. The starting page `./docs/index.rst` is in reStructuredText format, but all other pages are Markdown files. Please stick to Markdown, if possible.

When you have made changes to the documentation, build it. This can be done with *manati* itself from the project root directory:

```
manati run docs
```

Watch out for error messages in the console in case you have introduced some bugs to the documentation. A browser opens up with the newly built pages. If satisfied, commit your changes to your local git repository, push them to your remote repository, and send a pull request.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search