# How to contribute

Contributors are essential to Scapy (as they are to most open source

projects). Here is some advice to help you help the project!

## Project objectives

We try to keep Scapy as powerful as possible, to support as many

protocols and platforms as possible, to keep and make the code (and

the commit history) as clean as possible.

Since Scapy can be slow and memory consuming, we try to limit CPU and

memory usage, particularly in parts of the code often called.

## What to contribute

You want to spend time working on Scapy but have no (or little)

idea what to do? You can look for open issues

[labeled "contributions wanted"](https://github.com/secdev/scapy/labels/contributions%20wanted), or look at the [contributions roadmap](https://github.com/secdev/scapy/issues/399)

If you have any ideas of useful contributions that you cannot (or do

not want to) do yourself, open an issue and include

"contributions wanted" in the title.

Once you have chosen a contribution, open an issue to let other people

know you're working on it (or assign the existing issue to yourself)

and track your progress. You might want to ask whether you're working

in an appropriate direction, to avoid the frustration of seeing your

contribution rejected after a lot of work.

## Reporting issues

### Bugs

If you have installed Scapy through a package manager (from your Linux

or BSD system, from PyPI, etc.), please get and install the current

development code, and check that the bug still exists before

submitting an issue.

If you're not sure whether a behavior is a bug or not, submit an issue

and ask, don't be shy!

### Enhancements / feature requests

If you want a feature in Scapy, but cannot implement it yourself or

want some hints on how to do that, open an issue and include

"enhancement" in the title.

Explain if possible the API you would like to have (e.g., give examples

of function calls, packet creations, etc.).

## Submitting pull requests

### Coding style & conventions

- The code should be PEP-8 compliant; you can check your code with

[pep8](https://pypi.python.org/pypi/pep8) and the command `tox -e flake8`

- [Pylint](http://www.pylint.org/) can help you write good Python

code (even if respecting Pylint rules is sometimes either too hard

or even undesirable; human brain needed!).

- [Google Python Style Guide](https://google.github.io/styleguide/pyguide.html)

is a nice read!

- Avoid creating unnecessary `list` objects, particularly if they

can be huge (e.g., when possible, use `scapy.modules.six.range()` instead of

`range()`, `for line in fdesc` instead of `for line in

fdesc.readlines()`; more generally prefer generators over lists).

### Tests

Please consider adding tests for your new features or that trigger the

bug you are fixing. This will prevent a regression from being

unnoticed. Do not use the variable `\_` in your tests, as it could break them.

If you find yourself in a situation where your tests locally succeed but

fail if executed on the CI, try to enable the debugging option for the

dissector by setting `conf.debug\_dissector = 1`.

### New protocols

New protocols can go either in `scapy/layers` or to

`scapy/contrib`. Protocols in `scapy/layers` should be usually found

on common networks, while protocols in `scapy/contrib` should be

uncommon or specific.

To be precise, `scapy/layers` protocols should not be importing `scapy/contrib`

protocols, whereas `scapy/contrib` protocols may import both `scapy/contrib` and

`scapy/layers` protocols.

The detailed requirements are explained in [Design patterns](https://scapy.readthedocs.io/en/latest/build\_dissect.html#design-patterns) on Scapy's doc.

### Features

Protocol-related features should be implemented within the same module

as the protocol layers(s) (e.g., `traceroute()` is implemented in

`scapy/layers/inet.py`).

Other features may be implemented in a module (`scapy/modules`) or a

contribution (`scapy/contrib`).

### Core

If you contribute to Scapy's core (e.g., `scapy/base\_classes.py`,

`scapy/packet.py`, etc.), please be very careful with performances and

memory footprint, as it is easy to write Python code that wastes

memory or CPU cycles.

As an example, `Packet().\_\_init\_\_()` is called each time a \*\*layer\*\* is

parsed from a string (during a network capture or a PCAP file

read). Adding inefficient code here will have a disastrous effect on

Scapy's performances.

### Python 2 and 3 compatibility

The project aims to provide code that works both on Python 2 and Python 3. Therefore, some rules need to be applied to achieve compatibility:

- byte-string must be defined as `b"\x00\x01\x02"`

- exceptions must comply with the new Python 3 format: `except SomeError as e:`

- lambdas must be written using a single argument when using tuples: use `lambda x, y: x + f(y)` instead of `lambda (x, y): x + f(y)`.

- use int instead of long

- use list comprehension instead of map() and filter()

- use scapy.modules.six.moves.range instead of xrange and range

- use scapy.modules.six.itervalues(dict) instead of dict.values() or dict.itervalues()

- use scapy.modules.six.string\_types instead of basestring

- `\_\_bool\_\_ = \_\_nonzero\_\_` must be used when declaring `\_\_nonzero\_\_` methods

- `\_\_next\_\_ = next` must be used when declaring `next` methods in iterators

- `StopIteration` must NOT be used in generators (but it can still be used in iterators)

- `io.BytesIO` must be used instead of `StringIO` when using bytes

- `\_\_cmp\_\_` must not be used.

- UserDict should be imported via `six.UserDict`

### Code review

Maintainers tend to be picky, and you might feel frustrated that your

code (which is perfectly working in your use case) is not merged

faster.

Please don't be offended, and keep in mind that maintainers are

concerned about code maintainability and readability, commit history

(we use the history a lot, for example to find regressions or

understand why certain decisions have been made), performances,

integration in Scapy, API consistency (so that someone who knows how

to use Scapy will know how to use your code), etc.

\*\*Thanks for reading, happy hacking!\*\*