# HOW TO CONTRIBUTE TO TQDM

This file describes how to

- contribute changes to the project, and

- upload released to the pypi repository.

Most of the management commands have been directly placed inside the

Makefile:

```

make [<alias>] # on UNIX-like environments

python setup.py make [<alias>] # if make is unavailable

```

The latter depends on [`py-make>=0.1.0`](https://github.com/tqdm/py-make).

Use the alias `help` (or leave blank) to list all available aliases.

## HOW TO COMMIT CONTRIBUTIONS

Contributions to the project are made using the "Fork & Pull" model. The

typical steps would be:

1. create an account on [github](https://github.com)

2. fork [tqdm](https://github.com/tqdm/tqdm)

3. make a local clone: `git clone https://github.com/your\_account/tqdm.git`

4. make changes on the local copy

5. test (see below) and commit changes `git commit -a -m "my message"`

6. `push` to your GitHub account: `git push origin`

7. create a Pull Request (PR) from your GitHub fork

(go to your fork's webpage and click on "Pull Request."

You can then add a message to describe your proposal.)

## WHAT CODE LAYOUT SHOULD I FOLLOW?

Don't worry too much - maintainers can help reorganise contributions.

However it would be helpful to bear in mind:

- The standard core of `tqdm`, i.e. [`tqdm.std.tqdm`](tqdm/std.py)

+ must have no dependencies apart from pure python built-in standard libraries

+ must have negligible impact on performance

+ should have 100% coverage by unit tests

+ should be appropriately commented

+ should have well-formatted docstrings for functions

\* under 76 chars (incl. initial spaces) to avoid linebreaks in terminal pagers

\* use two spaces between variable name and colon, specify a type, and most likely state that it's optional: `VAR<space><space>:<space>TYPE[, optional]`

\* use [default: ...] for default values of keyword arguments

+ will not break backward compatibility unless there is a very good reason

\* e.g. breaking py26 compatibility purely in favour of readability (such as converting `dict(a=1)` to `{'a': 1}`) is not a good enough reason

+ API changes should be discussed carefully

+ remember, with millions of downloads per month, `tqdm` must be extremely fast and reliable

- Any other kind of change may be included in a (possibly new) submodule

+ submodules are likely single python files under the main [tqdm/](tqdm/) directory

\* large submodules requiring a sub-folder should be included in [`MANIFEST.in`](MANIFEST.in)

+ submodules extending `tqdm.std.tqdm` or any other module (e.g. [`tqdm.notebook.tqdm`](tqdm/notebook.py), [`tqdm.gui.tqdm`](tqdm/gui.py))

+ CLI wrapper `tqdm.cli`

\* if a newly added `tqdm.std.tqdm` option is not supported by the CLI, append to `tqdm.cli.UNSUPPORTED\_OPTS`

+ can implement anything from experimental new features to support for third-party libraries such as `pandas`, `numpy`, etc.

+ submodule maturity

\* alpha: experimental; missing unit tests, comments, and/or feedback; raises `tqdm.TqdmExperimentalWarning`

\* beta: well-used; commented, perhaps still missing tests

\* stable: >10 users; commented, 80% coverage

- `.meta/`

+ A "hidden" folder containing helper utilities not strictly part of `tqdm` distribution itself

## TESTING

Once again, don't worry too much - tests are automated online, and maintainers

can also help.

To test functionality (such as before submitting a Pull

Request), there are a number of unit tests.

### Standard unit tests

The standard way to run the tests:

- install `tox`

- `cd` to the root of the `tqdm` directory (in the same folder as this file)

- run the following command:

```

[python setup.py] make test

# or:

tox --skip-missing-interpreters

```

This will build the module and run the tests in a virtual environment.

Errors and coverage rates will be output to the console/log. (Ignore missing

interpreters errors - these are due to the local machine missing certain

versions of Python.)

Note: to install all versions of the Python interpreter that are specified

in [tox.ini](https://raw.githubusercontent.com/tqdm/tqdm/master/tox.ini),

you can use `MiniConda` to install a minimal setup. You must also make sure

that each distribution has an alias to call the Python interpreter:

`python27` for Python 2.7's interpreter, `python32` for Python 3.2's, etc.

### Alternative unit tests with Nose

Alternatively, use `nose` to run the tests just for the current Python version:

- install `nose` and `flake8`

- run the following command:

```

[python setup.py] make alltests

```

# MANAGE A NEW RELEASE

This section is intended for the project's maintainers and describes

how to build and upload a new release. Once again,

`[python setup.py] make [<alias>]` will help.

Also consider `pip install`ing development utilities:

`-r requirements-dev.txt` or `tqdm[dev]`.

## Pre-commit Hook

It's probably a good idea to add `[python setup.py] make pre-commit` to

`.git/hooks/pre-commit` for convenient local sanity-checking.

## Semantic Versioning

The tqdm repository managers should:

- regularly bump the version number in the file

[\_version.py](https://raw.githubusercontent.com/tqdm/tqdm/master/tqdm/\_version.py)

- follow the [Semantic Versioning](https://semver.org/) convention

- take care of this (instead of users) to avoid PR conflicts

solely due to the version file bumping

Note: tools can be used to automate this process, such as

[bumpversion](https://github.com/peritus/bumpversion) or

[python-semanticversion](https://github.com/rbarrois/python-semanticversion/).

## Checking setup.py

To check that the `setup.py` file is compliant with PyPI requirements (e.g.

version number; reStructuredText in `README.rst`) use:

```

[python setup.py] make testsetup

```

To upload just metadata (including overwriting mistakenly uploaded metadata)

to PyPI, use:

```

[python setup.py] make pypimeta

```

## Merging Pull Requests

This section describes how to cleanly merge PRs.

### 1 Rebase

From your project repository, merge and test

(replace `pr-branch-name` as appropriate):

```

git fetch origin

git checkout -b pr-branch-name origin/pr-branch-name

git rebase master

```

If there are conflicts:

```

git mergetool

git rebase --continue

```

### 2 Push

Update branch with the rebased history:

```

git push origin pr-branch-name --force

```

Non maintainers can stop here.

Note: NEVER just `git push --force` (this will push all local branches,

overwriting remotes).

### 3 Merge

```

git checkout master

git merge --no-ff pr-branch-name

```

### 4 Test

```

[python setup.py] make alltests

```

### 5 Version

Modify `tqdm/\_version.py` and amend the last (merge) commit:

```

git add tqdm/\_version.py

git commit --amend # Add "+ bump version" in the commit message

```

### 6 Push to master

```

git push origin master

```

## Building a Release and Uploading to PyPI

Formally publishing requires additional steps: testing and tagging.

### Test

- ensure that all online CI tests have passed

- check `setup.py` and `MANIFEST.in` - which define the packaging

process and info that will be uploaded to [PyPI](https://pypi.org) -

using `[python setup.py] make installdev`

### Tag

- ensure the version has been bumped, committed \*\*and\*\* tagged.

The tag format is `v{major}.{minor}.{patch}`, for example: `v4.4.1`.

The current commit's tag is used in the version checking process.

If the current commit is not tagged appropriately, the version will

display as `v{major}.{minor}.{patch}-{commit\_hash}`.

### Upload

Travis CI should automatically do this after pushing tags.

Manual instructions are given below in case of failure.

Build `tqdm` into a distributable python package:

```

[python setup.py] make build

```

This will generate several builds in the `dist/` folder. On non-windows

machines the windows `exe` installer may fail to build. This is normal.

Finally, upload everything to pypi. This can be done easily using the

[twine](https://github.com/pypa/twine) module:

```

[python setup.py] make pypi

```

Also, the new release can (should) be added to GitHub by creating a new

release from the [web interface](https://github.com/tqdm/tqdm/releases);

uploading packages from the `dist/` folder

created by `[python setup.py] make build`.

The [wiki] can be automatically updated with GitHub release notes by

running `make` within the wiki repository.

[wiki]: https://github.com/tqdm/tqdm/wiki

Docker images may be uploaded to <https://hub.docker.com/r/tqdm/tqdm>.

Assuming `docker` is

[installed](https://docs.docker.com/install/linux/docker-ce/ubuntu/):

```

make -B docker

docker login

docker push tqdm/tqdm:latest

docker push tqdm/tqdm:$(docker run -i --rm tqdm/tqdm -v)

```

Snaps may be uploaded to <https://snapcraft.io/tqdm>.

Assuming `snapcraft` is installed (`snap install snapcraft --classic --beta`):

```

make snap

snapcraft login

snapcraft push tqdm\*.snap --release stable

```

### Notes

- you can also test on the PyPI test servers `test.pypi.org`

before the real deployment

- in case of a mistake, you can delete an uploaded release on PyPI, but you

cannot re-upload another with the same version number

- in case of a mistake in the metadata on PyPI (e.g. bad README),

updating just the metadata is possible: `[python setup.py] make pypimeta`

## Updating Websites

The most important file is `.readme.rst`, which should always be kept up-to-date

and in sync with the in-line source documentation. This will affect all of the

following:

- `README.rst` (generated by `mkdocs.py` during `make build`)

- The [main repository site](https://github.com/tqdm/tqdm) which automatically

serves the latest `README.rst` as well as links to all of GitHub's features.

This is the preferred online referral link for `tqdm`.

- The [PyPI mirror](https://pypi.org/project/tqdm) which automatically

serves the latest release built from `README.rst` as well as links to past

releases.

- Many external web crawlers.

Additionally (less maintained), there exists:

- A [wiki] which is publicly editable.

- The [gh-pages project] which is built from the

[gh-pages branch](https://github.com/tqdm/tqdm/tree/gh-pages), which is

built using [asv](https://github.com/airspeed-velocity/asv).

- The [gh-pages root] which is built from a separate

[github.io repo](https://github.com/tqdm/tqdm.github.io).

[gh-pages project]: https://tqdm.github.io/tqdm/

[gh-pages root]: https://tqdm.github.io/

## Helper Bots

There are some helpers in

[.github/workflows](https://github.com/tqdm/tqdm/tree/master/.github/workflows)

to assist with maintenance.

- Comment Bot

+ allows maintainers to write `/tag vM.m.p commit\_hash` in an issue/PR to create a tag

- Post Release

+ automatically updates the [wiki]

+ automatically updates the [gh-pages root]

- Benchmark

+ automatically updates the [gh-pages project]

## QUICK DEV SUMMARY

For experienced devs, once happy with local master, follow the steps below.

Much is automated so really it's steps 1-6, then 12(a).

1. bump version in `tqdm/\_version.py`

2. test (`[python setup.py] make alltests`)

3. `git commit [--amend] # -m "bump version"`

4. `git push`

5. wait for tests to pass

a) in case of failure, fix and go back to (2)

6. `git tag vM.m.p && git push --tags` or comment `/tag vM.m.p commit\_hash`

7. \*\*`[AUTO:TravisCI]`\*\* `[python setup.py] make distclean`

8. \*\*`[AUTO:TravisCI]`\*\* `[python setup.py] make build`

9. \*\*`[AUTO:TravisCI]`\*\* upload to PyPI. either:

a) `[python setup.py] make pypi`, or

b) `twine upload -s -i $(git config user.signingkey) dist/tqdm-\*`

10. \*\*`[AUTO:TravisCI]`\*\* upload to docker hub:

a) `make -B docker`

b) `docker push tqdm/tqdm:latest`

c) `docker push tqdm/tqdm:$(docker run -i --rm tqdm/tqdm -v)`

11. \*\*`[AUTO:TravisCI]`\*\* upload to snapcraft:

a) `make snap`, and

b) `snapcraft push tqdm\*.snap --release stable`

12. Wait for travis to draft a new release on <https://github.com/tqdm/tqdm/releases>

a) replace the commit history with helpful release notes, and click publish

b) \*\*`[AUTO:TravisCI]`\*\* attach `dist/tqdm-\*` binaries

(usually only `\*.whl\*`)

13. \*\*`[SUB][AUTO:GHActions]`\*\* run `make` in the `wiki` submodule to update release notes

14. \*\*`[SUB][AUTO:GHActions]`\*\* run `make deploy` in the `docs` submodule to update website

15. \*\*`[SUB][AUTO:GHActions]`\*\* accept the automated PR in the `feedstock` submodule to update conda

16. \*\*`[AUTO:GHActions]`\*\* update the [gh-pages project] benchmarks

a) `[python setup.py] make testasvfull`

b) `asv gh-pages`

Key:

- \*\*`[AUTO:TravisCI]`\*\*: Travis CI should automatically do this after `git push --tags` (6)

- \*\*`[AUTO:GHActions]`\*\*: GitHub Actions CI should automatically do this after release (12a)

- \*\*`[SUB]`\*\*: Requires one-time `make submodules` to clone `docs`, `wiki`, and `feedstock`