

---

# **Python Developer's Guide Documentation**

**Brett Cannon**

**Sep 10, 2021**



# CONTENTS

<b>1</b>	<b>Quick Reference</b>	<b>3</b>
<b>2</b>	<b>Quick Links</b>	<b>5</b>
<b>3</b>	<b>Status of Python branches</b>	<b>7</b>
<b>4</b>	<b>Contributing</b>	<b>9</b>
<b>5</b>	<b>Proposing changes to Python itself</b>	<b>11</b>
<b>6</b>	<b>Other Interpreter Implementations</b>	<b>13</b>
<b>7</b>	<b>Key Resources</b>	<b>15</b>
<b>8</b>	<b>Additional Resources</b>	<b>17</b>
<b>9</b>	<b>Code of Conduct</b>	<b>19</b>
<b>10</b>	<b>Full Table of Contents</b>	<b>21</b>
10.1	Getting Started . . . . .	21
10.1.1	Install <code>git</code> . . . . .	21
10.1.2	Get the source code . . . . .	21
10.1.3	Compile and build . . . . .	22
10.1.4	Install dependencies . . . . .	25
10.1.5	Regenerate <code>configure</code> . . . . .	27
10.1.6	Troubleshoot the build . . . . .	27
10.1.7	Editors and Tools . . . . .	28
10.1.8	Directory structure . . . . .	28
10.2	Where to Get Help . . . . .	28
10.2.1	Discourse . . . . .	29
10.2.2	Mailing Lists . . . . .	29
10.2.3	Ask <code>#python-dev</code> . . . . .	29
10.2.4	Zulip . . . . .	29
10.2.5	Core Mentorship . . . . .	29
10.2.6	Core Developers Office Hours . . . . .	30
10.2.7	File a Bug . . . . .	30
10.3	Lifecycle of a Pull Request . . . . .	30
10.3.1	Introduction . . . . .	30
10.3.2	Quick Guide . . . . .	30
10.3.3	Step-by-step Guide . . . . .	31
10.3.4	Making Good PRs . . . . .	32

10.3.5	patchcheck	33
10.3.6	Making Good Commits	34
10.3.7	Licensing	34
10.3.8	Submitting	35
10.3.9	Converting an Existing Patch from b.p.o to GitHub	35
10.3.10	Reviewing	35
10.3.11	Leaving a Pull Request Review on GitHub	36
10.3.12	Dismissing Review from Another Core Developer	36
10.3.13	Committing/Rejecting	37
10.3.14	Crediting	37
10.4	Running & Writing Tests	37
10.4.1	Running	37
10.4.2	Writing	38
10.4.3	Benchmarks	39
10.5	Increase Test Coverage	39
10.5.1	Common Gotchas	39
10.5.2	Measuring Coverage	39
10.5.3	Filing the Issue	42
10.5.4	Measuring coverage of C code with gcov and lcov	42
10.6	Helping with Documentation	43
10.6.1	Python Documentation	43
10.6.2	Helping with documentation issues	44
10.6.3	Proofreading	44
10.6.4	Helping with the Developer's Guide	44
10.6.5	Developer's Guide workflow	44
10.7	Documenting Python	45
10.7.1	Introduction	45
10.7.2	Style guide	45
10.7.3	reStructuredText Primer	48
10.7.4	Additional Markup Constructs	52
10.7.5	Building the documentation	66
10.7.6	Translating	67
10.8	Silence Warnings From the Test Suite	69
10.9	Fixing "easy" Issues (and Beyond)	69
10.10	Issue Tracking	70
10.10.1	Using the Issue Tracker	70
10.10.2	Disagreement With a Resolution on the Issue Tracker	71
10.10.3	Helping Triage Issues	71
10.10.4	Gaining the "Developer" Role on the Issue Tracker	72
10.10.5	The Meta Tracker	72
10.11	Triaging an Issue	73
10.11.1	Python triage team	73
10.11.2	Becoming a member of the Python triage team	74
10.11.3	Fields in the Issue Tracker	75
10.11.4	Generating Special Links in a Comment	80
10.11.5	Checklist for Triaging	80
10.12	Following Python's Development	81
10.12.1	Mailing Lists	81
10.12.2	Discourse	82
10.12.3	IRC	82
10.12.4	Blogs	82
10.12.5	Standards of behaviour in these communication channels	82
10.12.6	Setting Expectations for Open Source Participation	82
10.12.7	Additional Repositories	82

10.13	Porting Python to a new platform . . . . .	82
10.14	How to Become a Core Developer . . . . .	83
10.14.1	What it Takes . . . . .	83
10.14.2	What it Means . . . . .	83
10.14.3	Gaining Commit Privileges . . . . .	84
10.14.4	Responsibilities . . . . .	85
10.15	Developer Log . . . . .	85
10.15.1	Procedure for Granting or Dropping Access . . . . .	92
10.16	Accepting Pull Requests . . . . .	92
10.16.1	Assessing a pull request . . . . .	92
10.16.2	Updating NEWS and What's New in Python . . . . .	93
10.16.3	Working with Git . . . . .	94
10.17	Development Cycle . . . . .	95
10.17.1	Branches . . . . .	95
10.17.2	Stages . . . . .	97
10.17.3	Repository Administration . . . . .	98
10.18	Continuous Integration . . . . .	99
10.18.1	In case of trouble . . . . .	100
10.18.2	Buildbot failures on Pull Requests . . . . .	100
10.18.3	Checking results of automatic builds . . . . .	100
10.18.4	Stability . . . . .	101
10.18.5	Flags-dependent failures . . . . .	101
10.18.6	Ordering-dependent failures . . . . .	101
10.18.7	Transient failures . . . . .	102
10.18.8	Custom builders . . . . .	103
10.19	Adding to the Stdlib . . . . .	103
10.19.1	Adding to a pre-existing module . . . . .	103
10.19.2	Adding a new module . . . . .	104
10.20	Changing the Python Language . . . . .	105
10.20.1	What Qualifies . . . . .	105
10.20.2	PEP Process . . . . .	105
10.20.3	Suggesting new features and language changes . . . . .	106
10.21	Experts Index . . . . .	106
10.21.1	Stdlib . . . . .	107
10.21.2	Tools . . . . .	111
10.21.3	Platforms . . . . .	112
10.21.4	Miscellaneous . . . . .	112
10.21.5	Documentation Translations . . . . .	113
10.22	gdb Support . . . . .	113
10.22.1	gdb 7 and later . . . . .	113
10.22.2	gdb 6 and earlier . . . . .	120
10.22.3	Updating auto-load-safe-path to allow test_gdb to run . . . . .	120
10.23	Exploring CPython's Internals . . . . .	120
10.23.1	CPython Source Code Layout . . . . .	120
10.23.2	Additional References . . . . .	121
10.24	Changing CPython's Grammar . . . . .	122
10.24.1	Abstract . . . . .	122
10.24.2	Checklist . . . . .	122
10.25	Guide of CPython's Parser . . . . .	123
10.25.1	Abstract . . . . .	123
10.25.2	How PEG Parsers Work . . . . .	123
10.25.3	Syntax . . . . .	125
10.25.4	Pegen . . . . .	131
10.25.5	Testing . . . . .	136

10.25.6	Debugging generated parsers . . . . .	136
10.25.7	References . . . . .	137
10.26	Design of CPython's Compiler . . . . .	137
10.26.1	Abstract . . . . .	137
10.26.2	Parsing . . . . .	137
10.26.3	Abstract Syntax Trees (AST) . . . . .	138
10.26.4	Memory Management . . . . .	139
10.26.5	Source Code to AST . . . . .	139
10.26.6	Control Flow Graphs . . . . .	141
10.26.7	AST to CFG to Bytecode . . . . .	142
10.26.8	Introducing New Bytecode . . . . .	143
10.26.9	Code Objects . . . . .	144
10.26.10	Important Files . . . . .	144
10.26.11	Known Compiler-related Experiments . . . . .	145
10.26.12	References . . . . .	146
10.27	Design of CPython's Garbage Collector . . . . .	146
10.27.1	Abstract . . . . .	146
10.27.2	Memory layout and object structure . . . . .	147
10.27.3	Identifying reference cycles . . . . .	148
10.27.4	Destroying unreachable objects . . . . .	151
10.27.5	Optimization: generations . . . . .	151
10.27.6	Optimization: reusing fields to save memory . . . . .	152
10.27.7	Optimization: delay tracking containers . . . . .	153
10.28	Updating standard library extension modules . . . . .	154
10.29	Changing Python's C API . . . . .	154
10.29.1	The internal API . . . . .	154
10.29.2	Public C API . . . . .	155
10.29.3	Limited API . . . . .	155
10.30	Coverity Scan . . . . .	157
10.30.1	Access to analysis reports . . . . .	157
10.30.2	Building and uploading analysis . . . . .	157
10.30.3	Known limitations . . . . .	158
10.30.4	Modeling . . . . .	158
10.30.5	Workflow . . . . .	158
10.30.6	Contact . . . . .	159
10.31	Dynamic Analysis with Clang . . . . .	159
10.31.1	What is Clang? . . . . .	159
10.31.2	What are Sanitizers? . . . . .	159
10.31.3	Clang/LLVM Setup . . . . .	160
10.31.4	Python Build Setup . . . . .	161
10.32	Running a buildbot worker . . . . .	165
10.32.1	Preparing for buildbot worker setup . . . . .	165
10.32.2	Setting up the buildbot worker . . . . .	165
10.32.3	Buildbot worker operation . . . . .	168
10.32.4	Required Ports . . . . .	169
10.32.5	Required Resources . . . . .	169
10.32.6	Security Considerations . . . . .	169
10.33	Core Developer Motivations and Affiliations . . . . .	170
10.33.1	Published entries . . . . .	170
10.33.2	Goals of this page . . . . .	173
10.33.3	Limitations on scope . . . . .	174
10.34	Git Bootcamp and Cheat Sheet . . . . .	174
10.34.1	Forking CPython GitHub Repository . . . . .	175
10.34.2	Cloning a Forked CPython Repository . . . . .	175

10.34.3	Listing the Remote Repositories . . . . .	175
10.34.4	Setting Up Your Name and Email Address . . . . .	175
10.34.5	Enabling autocrlf on Windows . . . . .	176
10.34.6	Creating and Switching Branches . . . . .	176
10.34.7	Deleting Branches . . . . .	176
10.34.8	Renaming Branch . . . . .	177
10.34.9	Staging and Committing Files . . . . .	177
10.34.10	Reverting Changes . . . . .	177
10.34.11	Stashing Changes . . . . .	178
10.34.12	Committing Changes . . . . .	178
10.34.13	Pushing Changes . . . . .	178
10.34.14	Creating a Pull Request . . . . .	178
10.34.15	Updating your CPython Fork . . . . .	178
10.34.16	Applying a Patch from Mercurial to Git . . . . .	179
10.34.17	Downloading Other's Patches . . . . .	180
10.34.18	Accepting and Merging a Pull Request . . . . .	180
10.34.19	Backporting Merged Changes . . . . .	181
10.34.20	Editing a Pull Request Prior to Merging . . . . .	182
10.35	Appendix: Topics . . . . .	182
10.35.1	Basics for contributors . . . . .	182
10.35.2	Core developers . . . . .	183
10.35.3	Development workflow for contributors . . . . .	183
10.35.4	Documenting Python and style guide . . . . .	183
10.35.5	Issue tracking and triaging . . . . .	183
10.35.6	Language development in depth . . . . .	183
10.35.7	Testing and continuous integration . . . . .	184
<b>Bibliography</b>		<b>185</b>
<b>Index</b>		<b>187</b>





This guide is a comprehensive resource for *contributing* to Python – for both new and experienced contributors. It is *maintained* by the same community that maintains Python. We welcome your contributions to Python!



## QUICK REFERENCE

Here are the basic steps needed to get *set up* and contribute a patch. This is meant as a checklist, once you know the basics. For complete instructions please see the *setup guide*.

1. Install and set up *Git* and other dependencies (see the *Git Setup* page for detailed information).
2. Fork the *CPython* repository to your GitHub account and *get the source code* using:

```
git clone https://github.com/<your_username>/cpython
cd cpython
```

3. Build Python, on UNIX and Mac OS use:

```
./configure --with-pydebug && make -j
```

and on Windows use:

```
PCbuild\build.bat -e -d
```

See also *more detailed instructions*, *how to install and build dependencies*, and the platform-specific pages for *UNIX*, *Mac OS*, and *Windows*.

4. *Run the tests*:

```
./python -m test -j3
```

On *most* Mac OS X systems, replace `./python` with `./python.exe`. On Windows, use `python.bat`.

5. Create a new branch where your work for the issue will go, e.g.:

```
git checkout -b fix-issue-12345 main
```

If an issue does not already exist, please *create it*. Trivial issues (e.g. typo fixes) do not require any issue to be created.

6. Once you fixed the issue, run the tests, run `make patchcheck`, and if everything is ok, commit.
7. Push the branch on your fork on GitHub and *create a pull request*. Include the issue number using `bpo-NNNN` in the pull request description. For example:

```
bpo-12345: Fix some bug in spam module
```

8. Add a News entry into the `Misc/NEWS.d` directory as individual file. The news entry can be created by using *blurb-it*, or the *blurb* tool and its `blurb add` command. Please read more about *blurb* in *documentation*.

---

**Note:** First time contributors will need to sign the Contributor Licensing Agreement (CLA) as described in the [Licensing](#) section of this guide.

---

## QUICK LINKS

Here are some links that you probably will reference frequently while contributing to Python:

- [Issue tracker](#)
- [Buildbot status](#)
- [Where to Get Help](#)
- [PEPs](#) (Python Enhancement Proposals)
- [Git Bootcamp and Cheat Sheet](#)



## STATUS OF PYTHON BRANCHES

Branch	Schedule	Status	First release	End-of-life	Release manager
main	<a href="#">PEP 664</a>	features	<i>2022-10-03</i>	<i>2027-10</i>	Pablo Galindo Salgado
3.10	<a href="#">PEP 619</a>	prerelease	<i>2021-10-04</i>	<i>2026-10</i>	Pablo Galindo Salgado
3.9	<a href="#">PEP 596</a>	bugfix	2020-10-05	<i>2025-10</i>	Łukasz Langa
3.8	<a href="#">PEP 569</a>	security	2019-10-14	<i>2024-10</i>	Łukasz Langa
3.7	<a href="#">PEP 537</a>	security	2018-06-27	<i>2023-06-27</i>	Ned Deily
3.6	<a href="#">PEP 494</a>	security	2016-12-23	<i>2021-12-23</i>	Ned Deily

Dates in *italic* are scheduled and can be adjusted.

The main branch is currently the future Python 3.11, and is the only branch that accepts new features. The latest release for each Python version can be found on the [download page](#).

Status:

**features** new features, bugfixes, and security fixes are accepted.

**prerelease** feature fixes, bugfixes, and security fixes are accepted for the upcoming feature release.

**bugfix** bugfixes and security fixes are accepted, new binaries are still released. (Also called **maintenance** mode or **stable** release)

**security** only security fixes are accepted and no more binaries are released, but new source-only versions can be released

**end-of-life** release cycle is frozen; no further changes can be pushed to it.

See also the [Development Cycle](#) page for more information about branches.

By default, the end-of-life is scheduled 5 years after the first release, but can be adjusted by the release manager of each branch. All Python 2 versions have reached end-of-life.





## CONTRIBUTING

We encourage everyone to contribute to Python and that’s why we have put up this developer’s guide. If you still have questions after reviewing the material in this guide, then the [Core Python Mentorship](#) group is available to help guide new contributors through the process.

A number of individuals from the Python community have contributed to a series of excellent guides at [Open Source Guides](#).

Core developers and contributors alike will find the following guides useful:

- [How to Contribute to Open Source](#)
- [Building Welcoming Communities](#)

Guide for contributing to Python:

New Contributors	Documentarians	Triagers	Core Developers
<i>Getting Started</i>	<i>Helping with Documentation</i>	<i>Issue Tracking</i>	<i>How to Become a Core Developer</i>
<i>Where to Get Help</i>	<i>Documenting Python</i>	<i>Triaging an Issue</i>	<i>Developer Log</i>
<i>Lifecycle of a Pull Request</i>	<i>Style guide</i>	<i>Helping Triage Issues</i>	<i>Accepting Pull Requests</i>
<i>Running &amp; Writing Tests</i>	<i>reStructuredText Primer</i>	<i>Experts Index</i>	<i>Development Cycle</i>
<i>Fixing “easy” Issues (and Beyond)</i>	<i>Translating</i>		<i>Core Developer Motivations and Affiliations</i>
<i>Following Python’s Development</i>			<i>Core Developers Office Hours</i>
<i>Git Bootcamp and Cheat Sheet</i>			

Advanced tasks and topics for once you are comfortable:

- *Silence Warnings From the Test Suite*
- Fixing issues found by the *buildbots*
- *Coverity Scan*
- Helping out with reviewing open pull requests. See *how to review a Pull Request*.
- *Fixing “easy” Issues (and Beyond)*

It is **recommended** that the above documents be read as needed. New contributors will build understanding of the CPython workflow by reading the sections mentioned in this table. You can stop where you feel comfortable and begin contributing immediately without reading and understanding these documents all at once. If you do choose to skip

around within the documentation, be aware that it is written assuming preceding documentation has been read so you may find it necessary to backtrack to fill in missing concepts and terminology.

## **PROPOSING CHANGES TO PYTHON ITSELF**

Improving Python’s code, documentation and tests are ongoing tasks that are never going to be “finished”, as Python operates as part of an ever-evolving system of technology. An even more challenging ongoing task than these necessary maintenance activities is finding ways to make Python, in the form of the standard library and the language definition, an even better tool in a developer’s toolkit.

While these kinds of change are much rarer than those described above, they do happen and that process is also described as part of this guide:

- *Adding to the Stdlib*
- *Changing the Python Language*

