# Contributing to PHP

Anybody who programs in PHP can be a contributing member of the community that

develops and deploys it; the task of deploying PHP, documentation and associated

websites is a never ending one. With every release, or release candidate comes a

wave of work, which takes a lot of organization and co-ordination.

You don't need any special access to download, build, debug and begin submitting

PHP or PECL code, tests or documentation. Once you've followed this guide and

had several contributions accepted, commit privileges are often quickly granted.

## Index

\* [Pull requests](#pull-requests)

\* [Filing bugs](#filing-bugs)

\* [Feature requests](#feature-requests)

\* [Writing tests](#writing-tests)

\* [Writing documentation](#writing-documentation)

\* [Getting help](#getting-help)

\* [PHP source code directory structure](#php-source-code-directory-structure)

\* [PHP internals](#php-internals)

\* [PECL extensions](#pecl-extensions)

\* [Checklist for submitting contribution](#checklist-for-submitting-contribution)

\* [What happens after submitting contribution?](#what-happens-after-submitting-contribution)

\* [What happens when your contribution is applied?](#what-happens-when-your-contribution-is-applied)

\* [Git commit rules](#git-commit-rules)

\* [Copyright and license headers](#copyright-and-license-headers)

## Pull requests

PHP welcomes pull requests to [add tests](#writing-tests), fix bugs and to

implement RFCs. Please be sure to include tests as appropriate!

If you are fixing a bug, then please submit your PR against the lowest actively

supported branch of PHP that the bug affects (only green branches on

[the supported version page](https://www.php.net/supported-versions.php) are

supported). For example, at the time of writing, the lowest supported version is

PHP 7.3, which corresponds to the `PHP-7.3` branch in Git. Please also make sure

you add a link to the PR in the bug on [the bug tracker](https://bugs.php.net/).

Pull requests implementing RFCs should be submitted against `master`.

Pull requests should \*never\* be submitted against `PHP-x.y.z` branches, as these

are only used for release management.

If your pull request exhibits conflicts with the base branch, please resolve

them by using `git rebase` instead of `git merge`.

Fork the official PHP repository and send a pull request. A notification will be

sent to the pull request mailing list. Sending a note to PHP Internals list

(internals@lists.php.net) may help getting more feedback and quicker turnaround.

You can also add pull requests to [bug reports](https://bugs.php.net/).

Read [Git access page](https://www.php.net/git.php) for help on using Git to get

and build PHP source code. We recommend to look at our

[workflow](https://wiki.php.net/vcs/gitworkflow) and our

[FAQ](https://wiki.php.net/vcs/gitfaq).

## Filing bugs

Bugs can be filed on the [PHP bug tracker](https://bugs.php.net/). If this is

the first time you've filed a bug, we suggest reading the

[guide to reporting a bug](https://bugs.php.net/how-to-report.php).

Where possible, please include a self-contained reproduction case!

## Feature requests

Feature requests are generally submitted in the form of

[Requests for Comments (RFC)](https://wiki.php.net/rfc/howto), ideally

accompanied by [pull requests](#pull-requests). You can find the extremely large

list of RFCs that have been previously considered on the

[PHP Wiki](https://wiki.php.net/rfc).

To create a RFC, discuss it with the extension maintainer, and discuss it on the

development mailing list internals@lists.php.net. RFC Wiki accounts can be

requested on https://wiki.php.net/start?do=register. PHP extension maintainers

can be found in the [EXTENSIONS](/EXTENSIONS) file in the PHP source code

repository. Mailing list subscription is explained on the

[mailing lists page](https://www.php.net/mailing-lists.php).

You may also want to read

[The Mysterious PHP RFC Process](https://blogs.oracle.com/opal/entry/the\_mysterious\_php\_rfc\_process)

for additional notes on the best way to approach submitting an RFC.

## Writing tests

We love getting new tests! PHP is a huge project and improving code coverage is

a huge win for every PHP user.

[Our QA site includes a page detailing how to write test cases.](https://qa.php.net/write-test.php)

Submitting test scripts helps us to understand what functionality has changed.

It is important for the stability and maintainability of PHP that tests are

comprehensive.

Failure conditions of `zend\_parse\_parameters`, `ZEND\_PARSE\_PARAMETERS()` and

similar functions should not be tested. These parameter parsing APIs are already

extensively tested, and additional tests only complicate future modifications.

For newly created tests, a `--CREDITS--` section should no longer be included,

as test authorship is already accurately tracked by Git. If multiple authors

should be credited, the `Co-authored-by` tag in the commit message may be used.

## Writing documentation

There are two ways to contribute to the PHP manual. You can edit the manual and

send patches anonymously via [the online editor](https://edit.php.net/), or you

can check the XML source out from Subversion and edit that and build it

[per the instructions on the documentation site](http://doc.php.net/tutorial/).

Patches created that way should be sent to the

[documentation mailing list](mailto:phpdoc@lists.php.net).

## Getting help

If you are having trouble contributing to PHP, or just want to talk to a human

about what you're working on, you can contact us via the

[internals mailing list](mailto:internals@lists.php.net), or the

[documentation mailing list](mailto:phpdoc@lists.php.net) for documentation

issues.

Although not a formal channel, you can also find a number of core developers on

the #php.pecl channel on [EFnet](http://www.efnet.org/). Similarly, many

documentation writers can be found on #php.doc. Windows development IRC channel

is available at #winphp-dev on FreeNode.

## PHP source code directory structure

PHP source code also includes several files generated during development and

several parts where maintenance is happening upstream in their respective

locations.

```bash

<php-src>/

?? .git/ # Git configuration and source directory

?? TSRM/ # Thread Safe Resource Manager

?? Zend/ # Zend Engine

?? zend\_vm\_execute.h # Generated by `Zend/zend\_vm\_gen.php`

?? zend\_vm\_opcodes.c # Generated by `Zend/zend\_vm\_gen.php`

?? zend\_vm\_opcodes.h # Generated by `Zend/zend\_vm\_gen.php`

?? ...

?? appveyor/ # Appveyor CI service files

?? build/ # \*nix build system files

?? ax\_\*.m4 # https://github.com/autoconf-archive/autoconf-archive

?? config.guess # https://git.savannah.gnu.org/cgit/config.git

?? config.sub # https://git.savannah.gnu.org/cgit/config.git

?? libtool.m4 # https://git.savannah.gnu.org/cgit/libtool.git

?? ltmain.sh # https://git.savannah.gnu.org/cgit/libtool.git

?? shtool # https://www.gnu.org/software/shtool/

?? ...

?? docs/ # PHP internals and repository documentation

?? ext/ # PHP core extensions

?? bcmath/

?? libbcmath/ # Forked and maintained in php-src

?? ...

?? curl/

?? sync-constants.php # The curl symbols checker

?? ...

?? date/

?? lib/ # Bundled datetime library https://github.com/derickr/timelib

?? parse\_date.c # Generated by re2c 0.15.3

?? parse\_iso\_intervals.c # Generated by re2c 0.15.3

?? ...

?? ...

?? ffi/

?? ffi\_parser.c # Generated by https://github.com/dstogov/llk

?? ...

?? fileinfo/

?? libmagic/ # Modified libmagic https://github.com/file/file

?? data\_file.c # Generated by `ext/fileinfo/create\_data\_file.php`

?? libmagic.patch # Modifications patch from upstream libmagic

?? magicdata.patch # Modifications patch from upstream libmagic

?? ...

?? gd/

?? libgd/ # Bundled and modified GD library https://github.com/libgd/libgd

?? ...

?? mbstring/

?? libmbfl/ # Forked and maintained in php-src

?? unicode\_data.h # Generated by `ext/mbstring/ucgendat/ucgendat.php`

?? ...

?? pcre/

?? pcre2lib/ # https://www.pcre.org/

?? ...

?? pdo\_mysql/

?? php\_pdo\_mysql\_sqlstate.h # Generated by `ext/pdo\_mysql/get\_error\_codes.php`

?? ...

?? skeleton/ # Skeleton for developing new extensions with `ext/ext\_skel.php`

?? ...

?? standard/

?? html\_tables/

?? mappings/ # https://www.unicode.org/Public/MAPPINGS/

?? ...

?? credits\_ext.h # Generated by `scripts/dev/credits`

?? credits\_sapi.h # Generated by `scripts/dev/credits`

?? html\_tables.h # Generated by `ext/standard/html\_tables/html\_table\_gen.php`

?? ...

?? tokenizer/

?? tokenizer\_data.c # Generated by `ext/tokenizer/tokenizer\_data\_gen.sh`

?? ...

?? zend\_test # For testing internal APIs. Not needed for regular builds.

?? ...

?? zip/ # Bundled https://github.com/pierrejoye/php\_zip

?? ...

?? ...

?? main/ # Binding that ties extensions, SAPIs, and engine together

?? streams/ # Streams layer subsystem

?? php\_version.h # Generated by release managers using `configure`

?? ...

?? pear/ # PEAR installation

?? sapi/ # PHP SAPI modules

?? cli/

?? mime\_type\_map.h # Generated by `sapi/cli/generate\_mime\_type\_map.php`

?? ...

?? ...

?? scripts/ # php-config, phpize and internal development scripts

?? tests/ # Core features tests

?? travis/ # Travis CI service files

?? win32/ # Windows build system files

?? cp\_enc\_map.c # Generated by `win32/cp\_enc\_map\_gen.exe`

?? ...

?? ...

```

## PHP internals

For information on PHP internal C functions see

[References about Maintaining and Extending PHP](https://wiki.php.net/internals/references).

Various external resources can be found on the web. A standard printed reference

is the book "Extending and Embedding PHP" by Sara Golemon.

## PECL extensions

If you are fixing broken functionality in a [PECL](https://pecl.php.net)

extension then create a bug or identify an existing bug at

[bugs.php.net](https://bugs.php.net). A bug can be used to track the change

progress and prevent your changes getting lost in the PHP mail archives. Some

PECL extensions have their own bug tracker locations and different contributing

procedures.

If your change is large then create a

[Request for Comments (RFC)](https://wiki.php.net/rfc), discuss it with the

extension maintainer, and discuss it on the development mailing list

pecl-dev@lists.php.net depending on the extension. PECL mailing list

subscription is explained on the

[PECL support page](https://pecl.php.net/support.php).

Update any open bugs and add a link to the source of your change. Send the patch

or pointer to the bug to pecl-dev@lists.php.net. Also CC the extension

maintainer. Explain what has been changed by your patch. Test scripts should be

included.

## Checklist for submitting contribution

- Read [Coding standards](/CODING\_STANDARDS.md) before you start working.

- Update git source just before running your final `diff` and before testing.

- Add in-line comments and/or have external documentation ready. Use only

`/\* \*/` style comments, not `//`.

- Create test scripts for use with `make test`.

- Run `make test` to check your change doesn't break other features.

- Rebuild PHP with `--enable-debug` which will show some kinds of memory errors

and check the PHP and web server error logs after running your PHP tests.

- Rebuild PHP with `--enable-zts` to check your change compiles and operates

correctly in a thread safe PHP.

- Review the change once more just before submitting it.

## What happens after submitting contribution?

If your change is easy to review and obviously has no side-effects, it might be

committed relatively quickly.

Because PHP is a volunteer-driven effort more complex changes will require

patience on your side. If you do not receive feedback in a few days, consider

bumping. Before doing this think about these questions:

- Did I send the patch to the right mailing list?

- Did I review the mailing list archives to see if these kind of changes had

been discussed before?

- Did I explain my change clearly?

- Is my change too hard to review? Because of what factors?

## What happens when your contribution is applied?

Your name will likely be included in the Git commit log. If your change affects

end users, a brief description and your name might be added to the [NEWS](/NEWS)

file.

## Git commit rules

This section refers to contributors that have Git push access and make commit

changes themselves. We'll assume you're basically familiar with Git, but feel

free to post your questions on the mailing list. Please have a look at the more

detailed [information on Git](https://git-scm.com/).

PHP is developed through the efforts of a large number of people. Collaboration

is a Good Thing(tm), and Git lets us do this. Thus, following some basic rules

with regards to Git usage will:

\* Make everybody happier, especially those responsible for maintaining PHP

itself.

\* Keep the changes consistently well documented and easily trackable.

\* Prevent some of those 'Oops' moments.

\* Increase the general level of good will on planet Earth.

Having said that, here are the organizational rules:

1. Respect other people working on the project.

2. Discuss any significant changes on the list before committing and get

confirmation from the release manager for the given branch.

3. Look at [EXTENSIONS](/EXTENSIONS) file to see who is the primary maintainer

of the code you want to contribute to.

4. If you "strongly disagree" about something another person did, don't start

fighting publicly - take it up in private email.

5. If you don't know how to do something, ask first!

6. Test your changes before committing them. We mean it. Really. To do so use

`make test`.

7. For development use the `--enable-debug` switch to avoid memory leaks and the

`--enable-zts` switch to ensure your code handles TSRM correctly and doesn't

break for those who need that.

Currently we have the following branches in use:

| Branch | |

| --------- | --------- |

| master | Active development branch for PHP 8.0, which is open for backwards incompatible changes and major internal API changes. |

| PHP-7.4 | Is used to release the PHP 7.4.x series. This is a current stable version and is open for bugfixes only. |

| PHP-7.3 | Is used to release the PHP 7.3.x series. This is a current stable version and is open for bugfixes only. |

| PHP-7.2 | Is used to release the PHP 7.2.x series. This is an old stable version and is open for security fixes only. |

| PHP-7.1 | This branch is closed. |

| PHP-7.0 | This branch is closed. |

| PHP-5.6 | This branch is closed. |

| PHP-5.5 | This branch is closed. |

| PHP-5.4 | This branch is closed. |

| PHP-5.3 | This branch is closed. |

| PHP-5.2 | This branch is closed. |

| PHP-5.1 | This branch is closed. |

| PHP-4.4 | This branch is closed. |

| PHP-X.Y.Z | These branches are used for the release managers for tagging the releases, hence they are closed to the general public. |

The next few rules are more of a technical nature:

1. All non-security bugfix changes should first go to the lowest bugfix branch

(i.e. 7.3) and then get merged up to all other branches. All security fixes

should go to the lowest security fixes branch (i.e 7.2). If a change is not

needed for later branches (i.e. fixes for features which were dropped from

later branches) an empty merge should be done.

2. All news updates intended for public viewing, such as new features, bug

fixes, improvements, etc., should go into the NEWS file of \*any stable

release\* version with the given change. In other words, news about a bug fix

which went into PHP-5.4, PHP-5.5 and master should be noted in both

PHP-5.4/NEWS and PHP-5.5/NEWS but not master, which is not a public released

version yet.

3. Do not commit multiple files and dump all messages in one commit. If you

modified several unrelated files, commit each group separately and provide a

nice commit message for each one. See example below.

4. Do write your commit message in such a way that it makes sense even without

the corresponding diff. One should be able to look at it, and immediately

know what was modified. Definitely include the function name in the message

as shown below.

5. In your commit messages, keep each line shorter than 80 characters. And try

to align your lines vertically, if they wrap. It looks bad otherwise.

6. If you modified a function that is callable from PHP, prepend PHP to the

function name as shown below.

The format of the commit messages is pretty simple.

<max 79 characters short description>\n

\n

<long description, 79 chars per line>

\n

An Example from the git project (commit 2b34e486bc):

pack-objects: Fix compilation with NO\_PTHREDS

It looks like commit 99fb6e04 (pack-objects: convert to use parse\_options(),

2012-02-01) moved the #ifdef NO\_PTHREDS around but hasn't noticed that the

'arg' variable no longer is available.

If you fix some bugs, you should note the bug ID numbers in your commit message.

Bug ID should be prefixed by `#`.

Example:

Fixed bug #14016 (pgsql notice handler double free crash bug.)

When you change the NEWS file for a bug fix, then please keep the bugs sorted in

decreasing order under the fixed version.

You can use [gitweb](https://git.php.net/) to look at PHP Git repository in

various ways.

## Copyright and license headers

New source code files should include the following header block:

```c

/\*

+----------------------------------------------------------------------+

| Copyright (c) The PHP Group |

+----------------------------------------------------------------------+

| This source file is subject to version 3.01 of the PHP license, |

| that is bundled with this package in the file LICENSE, and is |

| available through the world-wide-web at the following url: |

| https://www.php.net/license/3\_01.txt |

| If you did not receive a copy of the PHP license and are unable to |

| obtain it through the world-wide-web, please send a note to |

| license@php.net so we can mail you a copy immediately. |

+----------------------------------------------------------------------+

| Author: |

+----------------------------------------------------------------------+

\*/

```

Thank you for contributing to PHP!