How to contribute to Symfony

Version: master

generated on April 21, 2019

How to contribute to Symfony (master)

This work is licensed under the “Attribution-Share Alike 3.0 Unported” license (*http://creativecommons.org/licenses/by-sa/3.0/*).

You are free **to share** (to copy, distribute and transmit the work), and **to remix** (to adapt the work) under the following conditions:

- **Attribution**: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Share Alike**: If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license. For any reuse or distribution, you must make clear to others the license terms of this work.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor SensioLabs shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

If you find typos or errors, feel free to report them by creating a ticket on the Symfony ticketing system (*http://github.com/symfony/symfony-docs/issues*). Based on tickets and users feedback, this book is continuously updated.

Contents at a Glance

Code of Conduct 4
Reporting Guidelines 6
CARE Team....8
Code of Conduct: Concrete Example Document....10
Reporting a Bug 11
Creating a Bug Reproducer....12
Proposing a Change 14
Maintenance....20
Symfony Core Team....22
Security Issues....26
Running Symfony Tests....30
Our Backward Compatibility Promise....32
Experimental Features....42
Coding Standards 43
Conventions 48
Git....51
Symfony Code License 52
Documentation Format....53
Symfony Documentation License....57
Contributing to the Documentation 59
Documentation Standards....65
Translations....69
The Release Process 70
Respectful Review Comments 73
Community Reviews 77
Mentoring 81
Speaker Mentoring....82
Other Resources 83
Diversity Initiative Governance....84

Chapter 1
Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnic origin, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

CoC Active Response Ensurers, or CARE, are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

CARE team members have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by CARE team members.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior *may be reported* by contacting the *CARE team members*. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The CARE team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

CARE team members who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by the *core team*.

Attribution

This Code of Conduct is adapted from the *Contributor Covenant*[1], version 1.4, available at *https://www.contributor-covenant.org/version/1/4/code-of-conduct.html*

Related Documents

- Reporting Guidelines
- CARE Team
- Code of Conduct: Concrete Example Document

1. `https://www.contributor-covenant.org`

Chapter 2
Reporting Guidelines

If you believe someone is violating the Code of Conduct we ask that you report it to the *CARE team* by emailing, Twitter, in person or any way you see fit.

All reports will be kept confidential. The privacy of everyone included in the report is of our highest concern. Second to privacy there is transparency. After every report we will determine if a public statement should be made. If that's the case, the identities of all victims, reporters, and the accused will remain confidential unless those individuals instruct us otherwise. The details of the incident may also be generalized.

If you believe anyone is in physical danger or doing something that is against the law, please notify appropriate emergency services first by calling the relevant local authorities. If you are unsure what service or agency is appropriate to contact, include this in your report and we will attempt to notify them.

In your report please include:

- Your contact info for follow-up contact.
- Names (legal, nicknames, or pseudonyms) of any individuals involved.
- If there were other witnesses besides you, please try to include them as well.
- When and where the incident occurred. Please be as specific as possible.
- Your description of what occurred.
- If there is a publicly available record (e.g. a mailing list archive or a public IRC or Slack log), please include a link and a screenshot.
- If you believe this incident is ongoing.
- Any other information you believe we should have.

What happens after you file a report?

You will receive a reply from the *CARE team* acknowledging receipt as soon as possible, but within 24 hours.

The team member receiving the report will immediately contact all or some other CARE team members to review the incident and determine:

- What happened.
- Whether this event constitutes a Code of Conduct violation.
- What kind of response is appropriate.

If this is determined to be an ongoing incident or a threat to physical safety, the team's immediate priority will be to protect everyone involved. This means we may delay an "official" response until we believe that the situation has ended and that everyone is physically safe.

Once the team has a complete account of the events, they will make a decision as to how to respond. Responses may include:

- Nothing (if we determine no Code of Conduct violation occurred).
- A private reprimand from the Code of Conduct response team to the individual(s) involved.
- An imposed vacation (i.e. asking someone to "take a week off" from a mailing list or Slack).
- A permanent or temporary ban from some or all Symfony conference/community spaces (events, meetings, mailing lists, IRC, Slack, etc.)
- A request to engage in mediation and/or an accountability plan.
- On a case by case basis, other actions may be possible but will usually be coordinated with the core team and the Symfony company.

We'll respond within one week to the person who filed the report with either a resolution or an explanation of why the situation is not yet resolved.

Once we've determined our final actions, we'll contact the original reporter to let them know what action (if any) we'll be taking. We'll take into account feedback from the reporter on the appropriateness of our response, but our response will be determined by what will be best for community safety.

The CARE team keeps a private record of all incidents. By default all reports are shared with the entire CARE team unless the reporter specifically asks to exclude specific CARE team members, in which case these CARE team members will not be included in any communication on the incidents as well as records created related to the incidents.

CARE team members are expected to inform the CARE team and the reporters in case of conflicts on interest and recuse themselves if this is deemed a problem.

Appealing the response

Only permanent resolutions (such as bans) may be appealed. To appeal a decision of the working group, contact the *CARE team* with your appeal and they will review the case.

Document origin

Reporting Guidelines derived from those of the *Stumptown Syndicate*[1] and the *Django Software Foundation*[2].

Adopted by *Symfony*[3] organizers on 21 February 2018.

1. http://stumptownsyndicate.org/code-of-conduct/reporting-guidelines/
2. https://www.djangoproject.com/conduct/reporting/
3. https://symfony.com

Chapter 3
CARE Team

Our Pledge

In the interest of fostering an open and welcoming environment, the CoC Active Response Ensurers, or CARE, pledge to ensure that the spirit of the *Code of Conduct* is respected. Our main priority is to ensure the safety of our community members. The second goal is to help educate the community as a whole to be aware of the CoC and how to help implement its spirit throughout the community. In case these goals conflict, we will prioritize safety of community members over all other goals.

If you think there is or has been a violation to the code of conduct please contact the CARE team or if you prefer contact only individual members of the CARE team.

Members

Here are all the members of the Code of Conduct CARE team (in alphabetic order). You can contact any of them directly using the contact details below or you can also contact all of them at once by emailing **care@symfony.com**:

- **Emilie Lorenzo**
  - *E-mail*: emilie.lorenzo [at] symfony.com
  - *Twitter*: *@EmilieLorenzo*[1]
  - *SymfonyConnect*: *emilielorenzo*[2]
- **Michelle Sanver**
  - *E-mail*: hello [at] michellesanver.com
  - *Twitter*: *@michellesanver*[3]
  - *SymfonyConnect*: *michellesanver*[4]

1. `https://twitter.com/EmilieLorenzo`
2. `https://connect.symfony.com/profile/emilielorenzo`
3. `https://twitter.com/michellesanver`
4. `https://connect.symfony.com/profile/michellesanver`

- **Tobias Nyholm**
    - *E-mail*: tobias.nyholm [at] gmail.com
    - *Twitter*: *@tobiasnyholm*[5]
    - *SymfonyConnect*: *tobias*[6]

About the CARE Team

The *Symfony project leader* appoints the CARE team with candidates they see fit. The CARE team will consist of at least 3 people. The team should be representing as many demographics as possible, ideally from different employers.

5. `https://twitter.com/tobiasnyholm`
6. `https://connect.symfony.com/profile/tobias`

Chapter 4
Code of Conduct: Concrete Example Document

This is a living document that serves to give concrete examples of unwanted behavior. These examples have all taken place somewhere in the PHP community in the past, and are clear code of conduct violations according to the Symfony code of conduct.

Concrete Examples

- Unwelcome comments regarding a person's lifestyle choices and practices, including those related to food, health, parenting, drugs, and employment;
- Deliberate misgendering or use of *dead names*[1] (The birth name of a person who has since changed their name, often a transgender person);
- Threats of violence like "The person that created this PR should be punched in the face";
- Incitement of violence towards any individual, including encouraging a person to commit suicide or to engage in self-harm (even as a joke);
- Sustained disruption of discussion;
- Pattern of inappropriate social contact, such as requesting/assuming inappropriate levels of intimacy with others;
- Continued one-on-one communication after requests to cease;
- Putting down people based on their technology choices or their work.

The original list is inspired and modified from *geek feminism*[2] and confirmed by experiences from PHPWomen.

1. `https://en.wiktionary.org/wiki/deadname`
2. `https://geekfeminism.org/about/code-of-conduct`

Chapter 5
Reporting a Bug

Whenever you find a bug in Symfony, we kindly ask you to report it. It helps us make a better Symfony.

If you think you've found a security issue, please use the special *procedure* instead.

Before submitting a bug:

- Double-check the official *documentation* to see if you're not misusing the framework;
- Ask for assistance on *Stack Overflow*[1], on the #support channel of *the Symfony Slack*[2] or on the #symfony *IRC channel*[3] if you're not sure if your issue really is a bug.

If your problem definitely looks like a bug, report it using the official bug *tracker*[4] and follow some basic rules:

- Use the title field to clearly describe the issue;
- Describe the steps needed to reproduce the bug with short code examples (providing a unit test that illustrates the bug is best);
- If the bug you experienced is not obvious or affects more than one layer, providing a simple failing unit test may not be sufficient. In this case, please *provide a reproducer*;
- Give as much detail as possible about your environment (OS, PHP version, Symfony version, enabled extensions, ...);
- If you want to provide a stack trace you got on an HTML page, be sure to provide the plain text version, which should appear at the bottom of the page. *Do not* provide it as a screenshot, since search engines will not be able to index the text inside them. Same goes for errors encountered in a terminal, do not take a screenshot, but copy/paste the contents. If the stack trace is long, consider enclosing it in a *<details> HTML tag*[5]. **Be wary that stack traces may contain sensitive information, and if it is the case, be sure to redact them prior to posting your stack trace.**
- *(optional)* Attach a *patch*.

1. https://stackoverflow.com/questions/tagged/symfony
2. https://symfony.com/slack-invite
3. https://symfony.com/irc
4. https://github.com/symfony/symfony/issues
5. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/details

Chapter 6

Creating a Bug Reproducer

The main Symfony code repository receives thousands of issues reports per year. Some of those issues are so obvious or easy to understand, that Symfony Core developers can fix them without any other information. However, other issues are much harder to understand because developers can't easily reproduce them in their computers. That's when we'll ask you to create a "bug reproducer", which is the minimum amount of code needed to make the bug appear when executed.

Reproducing Simple Bugs

If you are reporting a bug related to some Symfony component used outside the Symfony framework, it's enough to share a small PHP script that when executed shows the bug:

Listing 6-1

```
// First, run "composer require symfony/validator"
// Then, execute this file:
<?php
require_once __DIR__.'/vendor/autoload.php';
use Symfony\Component\Validator\Constraints;

$wrongUrl = 'http://example.com/exploit.html?<script>alert(1);</script>';
$urlValidator = new Constraints\UrlValidator();
$urlConstraint = new Constraints\Url();

// The URL is wrong, so var_dump() should display an error, but it displays
// "null" instead because there is no context to build a validator violation
var_dump($urlValidator->validate($wrongUrl, $urlConstraint));
```

Reproducing Complex Bugs

If the bug is related to the Symfony Framework or if it's too complex to create a PHP script, it's better to reproduce the bug by creating a new project. To do so:

1. Create a new project:

Listing 6-2

```
$ composer create-project symfony/skeleton bug_app
```

1. Add and commit the changes generated by Symfony.

2. Now you must add the minimum amount of code to reproduce the bug. This is the trickiest part and it's explained a bit more later.
3. Add and commit your changes.
4. Create a *new repository*[1] on GitHub (give it any name).
5. Follow the instructions on GitHub to add the `origin` remote to your local project and push it.
6. Add a comment in your original issue report to share the URL of your forked project (e.g. `https://github.com/YOUR-GITHUB-USERNAME/symfony_issue_23567`) and, if necessary, explain the steps to reproduce (e.g. "browse this URL", "fill in this data in the form and submit it", etc.)

Adding the Minimum Amount of Code Possible

The key to create a bug reproducer is to solely focus on the feature that you suspect is failing. For example, imagine that you suspect that the bug is related to a route definition. Then, after creating your project:

1. Don't edit any of the default Symfony configuration options.
2. Don't copy your original application code and don't use the same structure of controllers, actions, etc. as in your original application.
3. Create a small controller and add your routing definition that shows the bug.
4. Don't create or modify any other file.
5. Execute `composer require symfony/web-server-bundle` and use the `server:run` command to browse to the new route and see if the bug appears or not.
6. If you can see the bug, you're done and you can already share the code with us.
7. If you can't see the bug, you must keep making small changes. For example, if your original route was defined using XML, forget about the previous route and define the route using XML instead. Or maybe your application registers some event listeners and that's where the real bug is. In that case, add an event listener that's similar to your real app to see if you can find the bug.

In short, the idea is to keep adding small and incremental changes to a new project until you can reproduce the bug.

1. `https://github.com/new`

Chapter 7

Proposing a Change

A pull request, "PR" for short, is the best way to provide a bug fix or to propose enhancements to Symfony.

Step 1: Check existing Issues and Pull Requests

Before working on a change, check to see if someone else also raised the topic or maybe even started working on a PR by *searching on GitHub*[1].

If you are unsure or if you have any questions during this entire process, please ask your questions on the #contribs channel on *Symfony Slack*[2].

Step 2: Setup your Environment

Install the Software Stack

Before working on Symfony, setup a friendly environment with the following software:

- Git;
- PHP version 5.5.9 or above.

Configure Git

Set up your user information with your real name and a working email address:

Listing 7-1

```
$ git config --global user.name "Your Name"
$ git config --global user.email you@example.com
```

1. https://github.com/symfony/symfony/issues?q=+is%3Aopen+
2. https://symfony.com/slack-invite

If you are new to Git, you are highly recommended to read the excellent and free *ProGit*[3] book.

If your IDE creates configuration files inside the project's directory, you can use global `.gitignore` file (for all projects) or `.git/info/exclude` file (per project) to ignore them. See *GitHub's documentation*[4].

Windows users: when installing Git, the installer will ask what to do with line endings, and suggests replacing all LF with CRLF. This is the wrong setting if you wish to contribute to Symfony! Selecting the as-is method is your best choice, as Git will convert your line feeds to the ones in the repository. If you have already installed Git, you can check the value of this setting by typing:

Listing 7-2
```
$ git config core.autocrlf
```

This will return either "false", "input" or "true"; "true" and "false" being the wrong values. Change it to "input" by typing:

Listing 7-3
```
$ git config --global core.autocrlf input
```

Replace --global by --local if you want to set it only for the active repository

Get the Symfony Source Code

Get the Symfony source code:

- Create a *GitHub*[5] account and sign in;
- Fork the *Symfony repository*[6] (click on the "Fork" button);
- After the "forking action" has completed, clone your fork locally (this will create a `symfony` directory):

Listing 7-4
```
$ git clone git@github.com:USERNAME/symfony.git
```

- Add the upstream repository as a remote:

Listing 7-5
```
$ cd symfony
$ git remote add upstream git://github.com/symfony/symfony.git
```

Check that the current Tests Pass

Now that Symfony is installed, check that all unit tests pass for your environment as explained in the dedicated *document*.

If tests are failing, check on *Travis-CI*[7] if the same test is failing there as well. In that case you do not need to be concerned about the test failing locally.

3. https://git-scm.com/book
4. https://help.github.com/articles/ignoring-files
5. https://github.com/join
6. https://github.com/symfony/symfony
7. https://travis-ci.org/symfony/symfony

Step 3: Work on your Pull Request

The License

Before you start, you should be aware that all the code you are going to submit must be released under the *MIT license*.

Choose the right Branch

Before working on a PR, you must determine on which branch you need to work:

- `3.4`, if you are fixing a bug for an existing feature or want to make a change that falls into the *list of acceptable changes in patch versions* (you may have to choose a higher branch if the feature you are fixing was introduced in a later version);

> - `master`, if you are adding a new feature.

All bug fixes merged into maintenance branches are also merged into more recent branches on a regular basis. For instance, if you submit a PR for the `3.4` branch, the PR will also be applied by the core team on the `master` branch.

Create a Topic Branch

Each time you want to work on a PR for a bug or on an enhancement, create a topic branch:

Listing 7-6
```
1 $ git checkout -b BRANCH_NAME master
```

Or, if you want to provide a bug fix for the `3.4` branch, first track the remote `3.4` branch locally:

Listing 7-7
```
1 $ git checkout -t origin/3.4
```

Then create a new branch off the `3.4` branch to work on the bug fix:

Listing 7-8
```
1 $ git checkout -b BRANCH_NAME 3.4
```

Use a descriptive name for your branch (`ticket_XXX` where `XXX` is the ticket number is a good convention for bug fixes).

The above checkout commands automatically switch the code to the newly created branch (check the branch you are working on with `git branch`).

Use your Branch in an Existing Project

If you want to test your code in an existing project that uses `symfony/symfony` or Symfony components, you can use the `link` utility provided in the Git repository you cloned previously. This tool scans the `vendor/` directory of your project, finds Symfony packages it uses, and replaces them by symbolic links to the ones in the Git repository.

Listing 7-9
```
1  $ php link /path/to/your/project
```

Before running the `link` command, be sure that the dependencies of the project you want to debug are installed by running `composer install` inside it.

Work on your Pull Request

Work on the code as much as you want and commit as much as you want; but keep in mind the following:

- Read about the Symfony *conventions* and follow the coding *standards* (use `git diff --check` to check for trailing spaces -- also read the tip below);
- Add unit tests to prove that the bug is fixed or that the new feature actually works;
- Try hard to not break backward compatibility (if you must do so, try to provide a compatibility layer to support the old way) -- PRs that break backward compatibility have less chance to be merged;
- Do atomic and logically separate commits (use the power of `git rebase` to have a clean and logical history);
- Never fix coding standards in some existing code as it makes the code review more difficult;
- Write good commit messages (see the tip below).

When submitting pull requests, *fabbot*[8] checks your code for common typos and verifies that you are using the PHP coding standards as defined in *PSR-1*[9] and *PSR-2*[10].

A status is posted below the pull request description with a summary of any problems it detects or any Travis CI build failures.

A good commit message is composed of a summary (the first line), optionally followed by a blank line and a more detailed description. The summary should start with the Component you are working on in square brackets (`[DependencyInjection]`, `[FrameworkBundle]`, ...). Use a verb (`fixed ...`, `added ...`, ...) to start the summary and don't add a period at the end.

Prepare your Pull Request for Submission

When your PR is not about a bug fix (when you add a new feature or change an existing one for instance), it must also include the following:

- An explanation of the changes in the relevant `CHANGELOG` file(s) (the `[BC BREAK]` or the `[DEPRECATION]` prefix must be used when relevant);
- An explanation on how to upgrade an existing application in the relevant `UPGRADE` file(s) if the changes break backward compatibility or if you deprecate something that will ultimately break backward compatibility.

Step 4: Submit your Pull Request

Whenever you feel that your PR is ready for submission, follow the following steps.

Rebase your Pull Request

Before submitting your PR, update your branch (needed if it takes you a while to finish your changes):

8. https://fabbot.io
9. https://www.php-fig.org/psr/psr-1/
10. https://www.php-fig.org/psr/psr-2/

Listing 7-10

```
1 $ git checkout master
2 $ git fetch upstream
3 $ git merge upstream/master
4 $ git checkout BRANCH_NAME
5 $ git rebase master
```

Replace `master` with the branch you selected previously (e.g. `3.4`) if you are working on a bug fix.

When doing the `rebase` command, you might have to fix merge conflicts. `git status` will show you the *unmerged* files. Resolve all the conflicts, then continue the rebase:

Listing 7-11

```
1 $ git add ... # add resolved files
2 $ git rebase --continue
```

Check that all tests still pass and push your branch remotely:

Listing 7-12

```
1 $ git push --force origin BRANCH_NAME
```

Make a Pull Request

You can now make a pull request on the `symfony/symfony` GitHub repository.

Take care to point your pull request towards `symfony:3.4` if you want the core team to pull a bug fix based on the `3.4` branch.

To ease the core team work, always include the modified components in your pull request message, like in:

Listing 7-13

```
1 [Yaml] fixed something
2 [Form] [Validator] [FrameworkBundle] added something
```

The default pull request description contains a table which you must fill in with the appropriate answers. This ensures that contributions may be reviewed without needless feedback loops and that your contributions can be included into Symfony as quickly as possible.

Some answers to the questions trigger some more requirements:

- If you answer yes to "Bug fix?", check if the bug is already listed in the Symfony issues and reference it/them in "Fixed tickets";
- If you answer yes to "New feature?", you must submit a pull request to the documentation and reference it under the "Doc PR" section;
- If you answer yes to "BC breaks?", the PR must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;
- If you answer yes to "Deprecations?", the PR must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;
- If you answer no to "Tests pass", you must add an item to a todo-list with the actions that must be done to fix the tests;
- If the "license" is not MIT, just don't submit the pull request as it won't be accepted anyway.

If some of the previous requirements are not met, create a todo-list and add relevant items:

Listing 7-14

```
- [ ] fix the tests as they have not been updated yet
- [ ] submit changes to the documentation
- [ ] document the BC breaks
```

If the code is not finished yet because you don't have time to finish it or because you want early feedback on your work, add an item to todo-list:

Listing 7-15

```
- [ ] finish the code
- [ ] gather feedback for my changes
```

As long as you have items in the todo-list, please prefix the pull request title with "[WIP]". If you do not yet want to trigger the automated tests, you can also set the PR to *draft status*[11].

In the pull request description, give as much details as possible about your changes (don't hesitate to give code examples to illustrate your points). If your pull request is about adding a new feature or modifying an existing one, explain the rationale for the changes. The pull request description helps the code review and it serves as a reference when the code is merged (the pull request description and all its associated comments are part of the merge commit message).

In addition to this "code" pull request, you must also send a pull request to the *documentation repository*[12] to update the documentation when appropriate.

Step 5: Receiving Feedback

We ask all contributors to follow some *best practices* to ensure a constructive feedback process.

If you think someone fails to keep this advice in mind and you want another perspective, please join the #contribs channel on *Symfony Slack*[13]. If you receive feedback you find abusive please contact the *CARE team*.

The *core team* is responsible for deciding which PR gets merged, so their feedback is the most relevant. So do not feel pressured to refactor your code immediately when someone provides feedback.

Rework your Pull Request

Based on the feedback on the pull request, you might need to rework your PR. Before re-submitting the PR, rebase with `upstream/master` or `upstream/3.4`, don't merge; and force the push to the origin:

Listing 7-16

```
$ git rebase -f upstream/master
$ git push --force origin BRANCH_NAME
```

When doing a `push --force`, always specify the branch name explicitly to avoid messing other branches in the repo (`--force` tells Git that you really want to mess with things so do it carefully).

Moderators earlier asked you to "squash" your commits. This means you will convert many commits to one commit. This is no longer necessary today, because Symfony project uses a proprietary tool which automatically squashes all commits before merging.

11. https://help.github.com/en/articles/about-pull-requests#draft-pull-requests
12. https://github.com/symfony/symfony-docs
13. https://symfony.com/slack-invite

Chapter 8
Maintenance

During the lifetime of a minor version, new releases (patch versions) are published on a monthly basis. This document describes the boundaries of acceptable changes.

Bug fixes are accepted under the following conditions:

- The change does not break valid unit tests;
- New unit tests cover the bug fix;
- The current buggy behavior is not widely used as a "feature".

When documentation (or phpdoc) is not in sync with the code, code behavior should always be considered as being the correct one.

Besides bug fixes, other minor changes can be accepted in a patch version:

- **Performance improvement**: Performance improvement should only be accepted if the changes are local (located in one class) and only for algorithmic issues (any such patches must come with numbers that show a significant improvement on real-world code);
- **Newer versions of PHP/HHVM**: Fixes that add support for newer versions of PHP or HHVM are acceptable if they don't break the unit test suite;
- **Newer versions of popular OSes**: Fixes that add support for newer versions of popular OSes (Linux, MacOS and Windows) are acceptable if they don't break the unit test suite;
- **Translations**: Translation updates and additions are accepted;
- **External data**: Updates for external data included in Symfony can be updated (like ICU for instance);
- **Version updates for Composer dependencies**: Changing the minimal version of a dependency is possible, bumping to a major one or increasing PHP minimal version is not;
- **Coding standard and refactoring**: Coding standard fixes or code refactoring are not recommended but can be accepted for consistency with the existing code base, if they are not too invasive, and if merging them on master would not lead to complex branch merging;
- **Tests**: Tests that increase the code coverage can be added.

Anything not explicitly listed above should be done on the next minor or major version instead (aka the *master* branch). For instance, the following changes are never accepted in a patch version:

- **New features**;

- **Backward compatibility breaks**: Note that backward compatibility breaks can be done when fixing a security issue if it would not be possible to fix it otherwise;
- **Support for external platforms**: Adding support for new platforms (like Google App Engine) cannot be done in patch versions;
- **Exception messages**: Exception messages must not be changed as some automated systems might rely on them (even if this is not recommended);
- **Adding new Composer dependencies**;
- **Support for newer major versions of Composer dependencies**: Taking into account support for newer versions of an existing dependency is not acceptable.
- **Web design**: Changing the web design of built-in pages like exceptions, the toolbar or the profiler is not allowed.

This policy is designed to enable a continuous upgrade path that allows one to move forward with newest Symfony versions in the safest way. One should be able to move PHP versions, OS or Symfony versions almost independently. That's the reason why supporting the latest PHP versions or OS features is considered as bug fixes.

Chapter 9
Symfony Core Team

The **Symfony Core** team is the group of developers that determine the direction and evolution of the Symfony project. Their votes rule if the features and patches proposed by the community are approved or rejected.

All the Symfony Core members are long-time contributors with solid technical expertise and they have demonstrated a strong commitment to drive the project forward.

This document states the rules that govern the Symfony core team. These rules are effective upon publication of this document and all Symfony Core members must adhere to said rules and protocol.

Core Organization

Symfony Core members are divided into groups. Each member can only belong to one group at a time. The privileges granted to a group are automatically granted to all higher priority groups.

The Symfony Core groups, in descending order of priority, are as follows:

1. **Project Leader**

- Elects members in any other group;
- Merges pull requests in all Symfony repositories.

2. **Mergers Team**

- Merge pull requests for the component or components on which they have been granted privileges.

3. **Deciders Team**

- Decide to merge or reject a pull request.

In addition, there are other groups created to manage specific topics:

Security Team

- Manage the whole security process (triaging reported vulnerabilities, fixing the reported issues, coordinating the release of security fixes, etc.)

Documentation Team

- Manage the whole *symfony-docs repository*[1].

Active Core Members

- **Project Leader**:
  - **Fabien Potencier** (*fabpot*[2]).
- **Mergers Team** (`@symfony/mergers` on GitHub):
  - **Tobias Schultze** (*Tobion*[3]) can merge into the *Routing*[4], *OptionsResolver*[5] and *PropertyAccess*[6] components;
  - **Nicolas Grekas** (*nicolas-grekas*[7]) can merge into all components, bridges and bundles;
  - **Christophe Coevoet** (*stof*[8]) can merge into all components, bridges and bundles;
  - **Kévin Dunglas** (*dunglas*[9]) can merge into the *PropertyInfo*[10], the *Serializer*[11] and the *WebLink*[12] components;
  - **Jakub Zalas** (*jakzal*[13]) can merge into the *DomCrawler*[14] and *Intl*[15] components;
  - **Christian Flothmann** (*xabbuh*[16]) can merge into the *Yaml*[17], and *Form*[18] components;
  - **Javier Eguiluz** (*javiereguiluz*[19]) can merge into the *WebProfilerBundle*[20];
  - **Grégoire Pineau** (*lyrixx*[21]) can merge into the *Workflow*[22] component;
  - **Ryan Weaver** (*weaverryan*[23]) can merge into the *Security*[24] component and the *SecurityBundle*[25];
  - **Robin Chalas** (*chalasr*[26]) can merge into the *Console*[27] and *Security*[28] components and the *SecurityBundle*[29];
  - **Maxime Steinhausser** (*ogizanagi*[30]) can merge into *Config*[31], *Console*[32], *Form*[33], *Serializer*[34], *DependencyInjection*[35], and *HttpKernel*[36] components;

---

1. `https://github.com/symfony/symfony-docs`
2. `https://github.com/fabpot/`
3. `https://github.com/Tobion/`
4. `https://github.com/symfony/routing`
5. `https://github.com/symfony/options-resolver`
6. `https://github.com/symfony/property-access`
7. `https://github.com/nicolas-grekas/`
8. `https://github.com/stof/`
9. `https://github.com/dunglas/`
10. `https://github.com/symfony/property-info`
11. `https://github.com/symfony/serializer`
12. `https://github.com/symfony/web-link`
13. `https://github.com/jakzal/`
14. `https://github.com/symfony/dom-crawler`
15. `https://github.com/symfony/intl`
16. `https://github.com/xabbuh/`
17. `https://github.com/symfony/yaml`
18. `https://github.com/symfony/form`
19. `https://github.com/javiereguiluz/`
20. `https://github.com/symfony/web-profiler-bundle`
21. `https://github.com/lyrixx/`
22. `https://github.com/symfony/workflow`
23. `https://github.com/weaverryan/`
24. `https://github.com/symfony/security`
25. `https://github.com/symfony/security-bundle`
26. `https://github.com/chalasr/`
27. `https://github.com/symfony/console`
28. `https://github.com/symfony/security`
29. `https://github.com/symfony/security-bundle`
30. `https://github.com/ogizanagi/`
31. `https://github.com/symfony/config`
32. `https://github.com/symfony/console`
33. `https://github.com/symfony/form`
34. `https://github.com/symfony/serializer`

- **Tobias Nyholm** (*Nyholm*[37]) manages the official and contrib recipes repositories;
- **Samuel Rozé** (*sroze*[38]) can merge into the *Messenger*[39] component.

- **Deciders Team** (`@symfony/deciders` on GitHub):
  - **Jordi Boggiano** (*seldaek*[40]);
  - **Lukas Kahwe Smith** (*lsmith77*[41]).
- **Security Team** (`@symfony/security` on GitHub):
  - **Fabien Potencier** (*fabpot*[42]);
  - **Michael Cullum** (*michaelcullum*[43]).
- **Documentation Team** (`@symfony/team-symfony-docs` on GitHub):
  - **Fabien Potencier** (*fabpot*[44]);
  - **Ryan Weaver** (*weaverryan*[45]);
  - **Christian Flothmann** (*xabbuh*[46]);
  - **Wouter De Jong** (*wouterj*[47]);
  - **Jules Pietri** (*HeahDude*[48]);
  - **Javier Eguiluz** (*javiereguiluz*[49]).
  - **Oskar Stark** (*OskarStark*[50]).

Former Core Members

They are no longer part of the core team, but we are very grateful for all their Symfony contributions:

- **Bernhard Schussek** (*webmozart*[51]);
- **Abdellatif AitBoudad** (*aitboudad*[52]);
- **Romain Neutron**.

Core Membership Application

At present, new Symfony Core membership applications are not accepted.

Core Membership Revocation

A Symfony Core membership can be revoked for any of the following reasons:

- Refusal to follow the rules and policies stated in this document;
- Lack of activity for the past six months;

---

35. `https://github.com/symfony/dependency-injection`
36. `https://github.com/symfony/http-kernel`
37. `https://github.com/Nyholm`
38. `https://github.com/sroze`
39. `https://github.com/symfony/messenger`
40. `https://github.com/Seldaek/`
41. `https://github.com/lsmith77/`
42. `https://github.com/fabpot/`
43. `https://github.com/michaelcullum`
44. `https://github.com/fabpot/`
45. `https://github.com/weaverryan/`
46. `https://github.com/xabbuh/`
47. `https://github.com/wouterj`
48. `https://github.com/HeahDude`
49. `https://github.com/javiereguiluz/`
50. `https://github.com/OskarStark`
51. `https://github.com/webmozart/`
52. `https://github.com/aitboudad/`

- Willful negligence or intent to harm the Symfony project;
- Upon decision of the **Project Leader**.

Should new Symfony Core memberships be accepted in the future, revoked members must wait at least 12 months before re-applying.

Code Development Rules

Symfony project development is based on pull requests proposed by any member of the Symfony community. Pull request acceptance or rejection is decided based on the votes cast by the Symfony Core members.

Pull Request Voting Policy

- `-1` votes must always be justified by technical and objective reasons;
- `+1` votes do not require justification, unless there is at least one `-1` vote;
- Core members can change their votes as many times as they desire during the course of a pull request discussion;
- Core members are not allowed to vote on their own pull requests.

Pull Request Merging Policy

A pull request **can be merged** if:

- It is a minor change [1];
- Enough time was given for peer reviews (at least 2 days for "regular" pull requests, and 4 days for pull requests with "a significant impact");
- At least the component's **Merger** or two other Core members voted `+1` and no Core member voted `-1`.

Pull Request Merging Process

All code must be committed to the repository through pull requests, except for minor changes [1] which can be committed directly to the repository.

Mergers must always use the command-line `gh` tool provided by the **Project Leader** to merge the pull requests.

Release Policy

The **Project Leader** is also the release manager for every Symfony version.

Symfony Core Rules and Protocol Amendments

The rules described in this document may be amended at anytime at the discretion of the **Project Leader**. [53]

53.

[1] *(1, 2)* Minor changes comprise typos, DocBlock fixes, code standards violations, and minor CSS, JavaScript and HTML modifications.

Chapter 10

Security Issues

This document explains how Symfony security issues are handled by the Symfony core team (Symfony being the code hosted on the main `symfony/symfony` Git repository).

Reporting a Security Issue

If you think that you have found a security issue in Symfony, don't use the bug tracker and don't publish it publicly. Instead, all security issues must be sent to **security [at] symfony.com**. Emails sent to this address are forwarded to the Symfony core team private mailing-list.

Resolving Process

For each report, we first try to confirm the vulnerability. When it is confirmed, the core team works on a solution following these steps:

1. Send an acknowledgement to the reporter;
2. Work on a patch;
3. Get a CVE identifier from *mitre.org*[1];
4. Write a security announcement for the official Symfony *blog*[2] about the vulnerability. This post should contain the following information:

   - a title that always include the "Security release" string;
   - a description of the vulnerability;
   - the affected versions;
   - the possible exploits;
   - how to patch/upgrade/workaround affected applications;
   - the CVE identifier;
   - credits.

5. Send the patch and the announcement to the reporter for review;
6. Apply the patch to all maintained versions of Symfony;

1. `https://cveform.mitre.org/`
2. `https://symfony.com/blog/`

7. Package new versions for all affected versions;
8. Publish the post on the official Symfony *blog*[3] (it must also be added to the "*Security Advisories*[4]" category);
9. Update the public *security advisories database*[5] maintained by the FriendsOfPHP organization and which is used by the `security:check` command.

Releases that include security issues should not be done on Saturday or Sunday, except if the vulnerability has been publicly posted.

While we are working on a patch, please do not reveal the issue publicly.

The resolution takes anywhere between a couple of days to a month depending on its complexity and the coordination with the downstream projects (see next paragraph).

Collaborating with Downstream Open-Source Projects

As Symfony is used by many large Open-Source projects, we standardized the way the Symfony security team collaborates on security issues with downstream projects. The process works as follows:

1. After the Symfony security team has acknowledged a security issue, it immediately sends an email to the downstream project security teams to inform them of the issue;
2. The Symfony security team creates a private Git repository to ease the collaboration on the issue and access to this repository is given to the Symfony security team, to the Symfony contributors that are impacted by the issue, and to one representative of each downstream projects;
3. All people with access to the private repository work on a solution to solve the issue via pull requests, code reviews, and comments;
4. Once the fix is found, all involved projects collaborate to find the best date for a joint release (there is no guarantee that all releases will be at the same time but we will try hard to make them at about the same time). When the issue is not known to be exploited in the wild, a period of two weeks seems like a reasonable amount of time.

The list of downstream projects participating in this process is kept as small as possible in order to better manage the flow of confidential information prior to disclosure. As such, projects are included at the sole discretion of the Symfony security team.

As of today, the following projects have validated this process and are part of the downstream projects included in this process:

- Drupal (releases typically happen on Wednesdays)
- eZPublish

Issue Severity

In order to determine the severity of a security issue we take into account the complexity of any potential attack, the impact of the vulnerability and also how many projects it is likely to affect. This score out of 15 is then converted into a level of: Low, Medium, High, Critical, or Exceptional.

3. `https://symfony.com/blog/`
4. `https://symfony.com/blog/category/security-advisories`
5. `https://github.com/FriendsOfPHP/security-advisories`

Attack Complexity

Score of between 1 and 5 depending on how complex it is to exploit the vulnerability

- 4 - 5 Basic: attacker must follow a set of simple steps
- 2 - 3 Complex: attacker must follow non-intuitive steps with a high level of dependencies
- 1 - 2 High: A successful attack depends on conditions beyond the attacker's control. That is, a successful attack cannot be accomplished at will, but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component before a successful attack can be expected.

Impact

Scores from the following areas are added together to produce a score. The score for Impact is capped at 6. Each area is scored between 0 and 4.

- Integrity: Does this vulnerability cause non-public data to be accessible? If so, does the attacker have control over the data disclosed? (0-4)
- Disclosure: Can this exploit allow system data (or data handled by the system) to be compromised? If so, does the attacker have control over modification? (0-4)
- Code Execution: Does the vulnerability allow arbitrary code to be executed on an end-users system, or the server that it runs on? (0-4)
- Availability: Is the availability of a service or application affected? Is it reduced availability or total loss of availability of a service / application? Availability includes networked services (e.g., databases) or resources such as consumption of network bandwidth, processor cycles, or disk space. (0-4)

Affected Projects

Scores from the following areas are added together to produce a score. The score for Affected Projects is capped at 4.

- Will it affect some or all using a component? (1-2)
- Is the usage of the component that would cause such a thing already considered bad practice? (0-1)
- How common/popular is the component (e.g. Console vs HttpFoundation vs Lock)? (0-2)
- Are a number of well-known open source projects using Symfony affected that requires coordinated releases? (0-1)

Score Totals

- Attack Complexity: 1 - 5
- Impact: 1 - 6
- Affected Projects: 1 - 4

Severity levels

- Low: 1 - 5
- Medium: 6 - 10
- High: 11 - 12
- Critical: 13 - 14
- Exceptional: 15

Security Advisories

You can check your Symfony application for known security vulnerabilities using the `security:check` command (see *How to Check for Known Security Vulnerabilities in Your Dependencies*).

Check the *Security Advisories*[6] blog category for a list of all security vulnerabilities that were fixed in Symfony releases, starting from Symfony 1.0.0.

6. `https://symfony.com/blog/category/security-advisories`

Chapter 11

Running Symfony Tests

The Symfony project uses a third-party service which automatically runs tests for any submitted *patch*. If the new code breaks any test, the pull request will show an error message with a link to the full error details.

In any case, it's a good practice to run tests locally before submitting a *patch* for inclusion, to check that you have not broken anything.

Before Running the Tests

To run the Symfony test suite, install the external dependencies used during the tests, such as Doctrine, Twig and Monolog. To do so, *install Composer* and execute the following:

Listing 11-1
```
$ composer update
```

Running the Tests

Then, run the test suite from the Symfony root directory with the following command:

Listing 11-2
```
$ php ./phpunit symfony
```

The output should display `OK`. If not, read the reported errors to figure out what's going on and if the tests are broken because of the new code.

The entire Symfony suite can take up to several minutes to complete. If you want to test a single component, type its path after the `phpunit` command, e.g.:

Listing 11-3
```
$ php ./phpunit src/Symfony/Component/Finder/
```

On Windows, install the *Cmder*[1], *ConEmu*[2], *ANSICON*[3] or *Mintty*[4] free applications to see colored test results.

1. `http://cmder.net/`
2. `https://conemu.github.io/`
3. `https://github.com/adoxa/ansicon/releases`
4. `https://mintty.github.io/`

Chapter 12
Our Backward Compatibility Promise

Ensuring smooth upgrades of your projects is our first priority. That's why we promise you backward compatibility (BC) for all minor Symfony releases. You probably recognize this strategy as *Semantic Versioning*[1]. In short, Semantic Versioning means that only major releases (such as 2.0, 3.0 etc.) are allowed to break backward compatibility. Minor releases (such as 2.5, 2.6 etc.) may introduce new features, but must do so without breaking the existing API of that release branch (2.x in the previous example).

This promise was introduced with Symfony 2.3 and does not apply to previous versions of Symfony.

However, backward compatibility comes in many different flavors. In fact, almost every change that we make to the framework can potentially break an application. For example, if we add a new method to a class, this will break an application which extended this class and added the same method, but with a different method signature.

Also, not every BC break has the same impact on application code. While some BC breaks require you to make significant changes to your classes or your architecture, others are fixed by changing the name of a method.

That's why we created this page for you. The section "Using Symfony Code" will tell you how you can ensure that your application won't break completely when upgrading to a newer version of the same major release branch.

The second section, "Working on Symfony Code", is targeted at Symfony contributors. This section lists detailed rules that every contributor needs to follow to ensure smooth upgrades for our users.

Experimental Features and code marked with the `@internal` tags are excluded from our Backward Compatibility promise.

Also note that backward compatibility breaks are tolerated if they are required to fix a security issue.

1. `https://semver.org/`

Using Symfony Code

If you are using Symfony in your projects, the following guidelines will help you to ensure smooth upgrades to all future minor releases of your Symfony version.

Using our Interfaces

All interfaces shipped with Symfony can be used in type hints. You can also call any of the methods that they declare. We guarantee that we won't break code that sticks to these rules.

The exception to this rule are interfaces tagged with `@internal`. Such interfaces should not be used or implemented.

If you implement an interface, we promise that we won't ever break your code.

The following table explains in detail which use cases are covered by our backward compatibility promise:

Use Case	**Backward Compatibility**
If you...	**Then we guarantee BC...**
Type hint against the interface	Yes
Call a method	Yes
If you implement the interface and...	**Then we guarantee BC...**
Implement a method	Yes
Add an argument to an implemented method	Yes
Add a default value to an argument	Yes
Add a return type to an implemented method	Yes

Using our Classes

All classes provided by Symfony may be instantiated and accessed through their public methods and properties.

Classes, properties and methods that bear the tag `@internal` as well as the classes located in the various `*\\Tests\\` namespaces are an exception to this rule. They are meant for internal use only and should not be accessed by your own code.

To be on the safe side, check the following table to know which use cases are covered by our backward compatibility promise:

Use Case	**Backward Compatibility**
If you...	**Then we guarantee BC...**
Type hint against the class	Yes
Create a new instance	Yes
Extend the class	Yes
Access a public property	Yes

Use Case	Backward Compatibility
Call a public method	Yes
If you extend the class and...	**Then we guarantee BC...**
Access a protected property	Yes
Call a protected method	Yes
Override a public property	Yes
Override a protected property	Yes
Override a public method	Yes
Override a protected method	Yes
Add a new property	No
Add a new method	No
Add an argument to an overridden method	Yes
Add a default value to an argument	Yes
Call a private method (via Reflection)	No
Access a private property (via Reflection)	No

Using our Traits

All traits provided by Symfony may be used in your classes.

The exception to this rule are traits tagged with `@internal`. Such traits should not be used.

To be on the safe side, check the following table to know which use cases are covered by our backward compatibility promise:

Use Case	Backward Compatibility
If you...	**Then we guarantee BC...**
Use a trait	Yes
If you use the trait and...	**Then we guarantee BC...**
Use it to implement an interface	Yes
Use it to implement an abstract method	Yes
Use it to extend a parent class	Yes
Use it to define an abstract class	Yes
Use a public, protected or private property	Yes
Use a public, protected or private method	Yes

Working on Symfony Code

Do you want to help us improve Symfony? That's great! However, please stick to the rules listed below in order to ensure smooth upgrades for our users.

Changing Interfaces

This table tells you which changes you are allowed to do when working on Symfony's interfaces:

Type of Change	Change Allowed
Remove entirely	No
Change name or namespace	No
Add parent interface	Yes [2]
Remove parent interface	No
Methods	
Add method	No
Remove method	No
Change name	No
Move to parent interface	Yes
Add argument without a default value	No
Add argument with a default value	No
Remove argument	Yes [3]
Add default value to an argument	No
Remove default value of an argument	No
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Add return type	No
Remove return type	No [9]
Change return type	No
Static Methods	
Turn non static into static	No
Turn static into non static	No
Constants	
Add constant	Yes
Remove constant	No
Change value of a constant	Yes [1] [5]

Changing Classes

This table tells you which changes you are allowed to do when working on Symfony's classes:

Type of Change	Change Allowed
Remove entirely	No
Make final	No [6]
Make abstract	No
Change name or namespace	No
Change parent class	Yes [4]
Add interface	Yes
Remove interface	No
Public Properties	
Add public property	Yes
Remove public property	No
Reduce visibility	No
Move to parent class	Yes
Protected Properties	
Add protected property	Yes
Remove protected property	No [7]
Reduce visibility	No [7]
Make public	No [7]
Move to parent class	Yes
Private Properties	
Add private property	Yes
Make public or protected	Yes
Remove private property	Yes
Constructors	
Add constructor without mandatory arguments	Yes [1]
Remove constructor	No
Reduce visibility of a public constructor	No
Reduce visibility of a protected constructor	No [7]
Move to parent class	Yes
Destructors	
Add destructor	Yes
Remove destructor	No
Move to parent class	Yes
Public Methods	
Add public method	Yes
Remove public method	No
Change name	No

Type of Change	Change Allowed
Reduce visibility	No
Make final	No [6]
Move to parent class	Yes
Add argument without a default value	No
Add argument with a default value	No [7] [8]
Remove argument	Yes [3]
Add default value to an argument	No [7] [8]
Remove default value of an argument	No
Add type hint to an argument	No [7] [8]
Remove type hint of an argument	No [7] [8]
Change argument type	No [7] [8]
Add return type	No [7] [8]
Remove return type	No [7] [8] [9]
Change return type	No [7] [8]
Protected Methods	
Add protected method	Yes
Remove protected method	No [7]
Change name	No [7]
Reduce visibility	No [7]
Make final	No [6]
Make public	No [7] [8]
Move to parent class	Yes
Add argument without a default value	No [7]
Add argument with a default value	No [7] [8]
Remove argument	Yes [3]
Add default value to an argument	No [7] [8]
Remove default value of an argument	No [7]
Add type hint to an argument	No [7] [8]
Remove type hint of an argument	No [7] [8]
Change argument type	No [7] [8]
Add return type	No [7] [8]
Remove return type	No [7] [8] [9]
Change return type	No [7] [8]
Private Methods	
Add private method	Yes
Remove private method	Yes

Type of Change	Change Allowed
Change name	Yes
Make public or protected	Yes
Add argument without a default value	Yes
Add argument with a default value	Yes
Remove argument	Yes
Add default value to an argument	Yes
Remove default value of an argument	Yes
Add type hint to an argument	Yes
Remove type hint of an argument	Yes
Change argument type	Yes
Add return type	Yes
Remove return type	Yes
Change return type	Yes
Static Methods and Properties	
Turn non static into static	No [7] [8]
Turn static into non static	No
Constants	
Add constant	Yes
Remove constant	No
Change value of a constant	Yes [1] [5]

Changing Traits

This table tells you which changes you are allowed to do when working on Symfony's traits:

Type of Change	Change Allowed
Remove entirely	No
Change name or namespace	No
Use another trait	Yes
Public Properties	
Add public property	Yes
Remove public property	No
Reduce visibility	No
Move to a used trait	Yes
Protected Properties	
Add protected property	Yes
Remove protected property	No

Type of Change	Change Allowed
Reduce visibility	No
Make public	No
Move to a used trait	Yes
Private Properties	
Add private property	Yes
Remove private property	No
Make public or protected	Yes
Move to a used trait	Yes
Constructors and destructors	
Have constructor or destructor	No
Public Methods	
Add public method	Yes
Remove public method	No
Change name	No
Reduce visibility	No
Make final	No [6]
Move to used trait	Yes
Add argument without a default value	No
Add argument with a default value	No
Remove argument	No
Add default value to an argument	No
Remove default value of an argument	No
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Change return type	No
Protected Methods	
Add protected method	Yes
Remove protected method	No
Change name	No
Reduce visibility	No
Make final	No [6]
Make public	No [8]
Move to used trait	Yes
Add argument without a default value	No
Add argument with a default value	No

Type of Change	Change Allowed
Remove argument	No
Add default value to an argument	No
Remove default value of an argument	No
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Change return type	No
Private Methods	
Add private method	Yes
Remove private method	No
Change name	No
Make public or protected	Yes
Move to used trait	Yes
Add argument without a default value	No
Add argument with a default value	No
Remove argument	No
Add default value to an argument	No
Remove default value of an argument	No
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Add return type	No
Remove return type	No
Change return type	No
Static Methods and Properties	
Turn non static into static	No
Turn static into non static	No

2 3 4 5 6 7 8 9 10

2.

[1] *(1, 2, 3)* Should be avoided. When done, this change must be documented in the UPGRADE file.

3.

[2] The added parent interface must not introduce any new methods that don't exist in the interface already.

4.

[3] *(1, 2, 3)* Only the last argument(s) of a method may be removed, as PHP does not care about additional arguments that you pass to a method.

5.

[4] When changing the parent class, the original parent class must remain an ancestor of the class.

6.

[5] *(1, 2)* The value of a constant may only be changed when the constants aren't used in configuration (e.g. Yaml and XML files), as these do not support constants and have to hardcode the value. For instance, event name constants can't change the value without introducing a BC break. Additionally, if a constant will likely be used in objects that are serialized, the value of a constant should not be changed.

7.

[6] *(1, 2, 3, 4, 5)* Allowed using the `@final` annotation.

8.

[7] *(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27)* Allowed if the class is final. Classes that received the `@final` annotation after their first release are considered final in their next major version. Changing an argument type is only possible with a parent type. Changing a return type is only possible with a child type.

9.

[8] *(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19)* Allowed if the method is final. Methods that received the `@final` annotation after their first release are considered final in their next major version. Changing an argument type is only possible with a parent type. Changing a return type is only possible with a child type.

10.

[9] *(1, 2, 3)* Allowed for the `void` return type.

Chapter 13

Experimental Features

All Symfony features benefit from our *Backward Compatibility Promise* to give developers the confidence to upgrade to new versions safely and more often.

But sometimes, a new feature is controversial or you cannot find a convincing API. In such cases, we prefer to gather feedback from real-world usage, adapt the API, or remove it altogether. Doing so is not possible with a no BC-break approach.

To avoid being bound to our backward compatibility promise, such features can be marked as **experimental** and their classes and methods must be marked with the `@experimental` tag.

A feature can be marked as being experimental for only one minor version, and can never be introduced in an LTS version. The core team can decide to extend the experimental period for another minor version on a case by case basis.

To ease upgrading projects using experimental features, the changelog must explain backward incompatible changes and explain how to upgrade code.

Chapter 14
Coding Standards

Symfony code is contributed by thousands of developers around the world. To make every piece of code look and feel familiar, Symfony defines some coding standards that all contributions must follow.

These Symfony coding standards are based on the *PSR-1*[1], *PSR-2*[2] and *PSR-4*[3] standards, so you may already know most of them.

Making your Code Follow the Coding Standards

Instead of reviewing your code manually, Symfony makes it simple to ensure that your contributed code matches the expected code syntax. First, install the *PHP CS Fixer tool*[4] and then, run this command to fix any problem:

Listing 14-1
```
$ cd your-project/
$ php php-cs-fixer.phar fix -v
```

If you forget to run this command and make a pull request with any syntax issue, our automated tools will warn you about that and will provide the solution.

Symfony Coding Standards in Detail

If you want to learn about the Symfony coding standards in detail, here's a short example containing most features described below:

Listing 14-2
```
/*
 * This file is part of the Symfony package.
 *
 * (c) Fabien Potencier <fabien@symfony.com>
 *
 * For the full copyright and license information, please view the LICENSE
```

1. https://www.php-fig.org/psr/psr-1/
2. https://www.php-fig.org/psr/psr-2/
3. https://www.php-fig.org/psr/psr-4/
4. http://cs.sensiolabs.org/

```
7     * file that was distributed with this source code.
8     */
9
10   namespace Acme;
11
12   /**
13    * Coding standards demonstration.
14    */
15   class FooBar
16   {
17       const SOME_CONST = 42;
18
19       /**
20        * @var string
21        */
22       private $fooBar;
23
24       /**
25        * @param string $dummy Some argument description
26        */
27       public function __construct($dummy)
28       {
29           $this->fooBar = $this->transformText($dummy);
30       }
31
32       /**
33        * @return string
34        *
35        * @deprecated
36        */
37       public function someDeprecatedMethod()
38       {
39           @trigger_error(sprintf('The %s() method is deprecated since version 2.8 and will be removed in 3.0.
40   Use Acme\Baz::someMethod() instead.', __METHOD__), E_USER_DEPRECATED);
41
42           return Baz::someMethod();
43       }
44
45       /**
46        * Transforms the input given as first argument.
47        *
48        * @param bool|string $dummy   Some argument description
49        * @param array       $options An options collection to be used within the transformation
50        *
51        * @return string|null The transformed input
52        *
53        * @throws \RuntimeException When an invalid option is provided
54        */
55       private function transformText($dummy, array $options = [])
56       {
57           $defaultOptions = [
58               'some_default' => 'values',
59               'another_default' => 'more values',
60           ];
61
62           foreach ($options as $option) {
63               if (!in_array($option, $defaultOptions)) {
64                   throw new \RuntimeException(sprintf('Unrecognized option "%s"', $option));
65               }
66           }
67
68           $mergedOptions = array_merge(
69               $defaultOptions,
70               $options
71           );
72
73           if (true === $dummy) {
74               return null;
75           }
76
77           if ('string' === $dummy) {
```

```
78              if ('values' === $mergedOptions['some_default']) {
79                  return substr($dummy, 0, 5);
80              }
81
82              return ucwords($dummy);
83          }
84      }
85
86      /**
87       * Performs some basic check for a given value.
88       *
89       * @param mixed $value     Some value to check against
90       * @param bool  $theSwitch Some switch to control the method's flow
91       *
92       * @return bool|void The resultant check if $theSwitch isn't false, void otherwise
93       */
94      private function reverseBoolean($value = null, $theSwitch = false)
95      {
96          if (!$theSwitch) {
97              return;
98          }
99
100         return !$value;
101     }
    }
```

Structure

- Add a single space after each comma delimiter;
- Add a single space around binary operators (`==`, `&&`, ...), with the exception of the concatenation (`.`) operator;
- Place unary operators (`!`, `--`, ...) adjacent to the affected variable;
- Always use *identical comparison*[5] unless you need type juggling;
- Use *Yoda conditions*[6] when checking a variable against an expression to avoid an accidental assignment inside the condition statement (this applies to `==`, `!=`, `===`, and `!==`);
- Add a comma after each array item in a multi-line array, even after the last one;
- Add a blank line before `return` statements, unless the return is alone inside a statement-group (like an `if` statement);
- Use `return null;` when a function explicitly returns `null` values and use `return;` when the function returns `void` values;
- Use braces to indicate control structure body regardless of the number of statements it contains;
- Define one class per file - this does not apply to private helper classes that are not intended to be instantiated from the outside and thus are not concerned by the *PSR-0*[7] and *PSR-4*[8] autoload standards;
- Declare the class inheritance and all the implemented interfaces on the same line as the class name;
- Declare class properties before methods;
- Declare public methods first, then protected ones and finally private ones. The exceptions to this rule are the class constructor and the `setUp()` and `tearDown()` methods of PHPUnit tests, which must always be the first methods to increase readability;
- Declare all the arguments on the same line as the method/function name, no matter how many arguments there are;
- Use parentheses when instantiating classes regardless of the number of arguments the constructor has;
- Exception and error message strings must be concatenated using *`sprintf`*[9];

5. https://php.net/manual/en/language.operators.comparison.php
6. https://en.wikipedia.org/wiki/Yoda_conditions
7. https://www.php-fig.org/psr/psr-0/
8. https://www.php-fig.org/psr/psr-4/
9. https://secure.php.net/manual/en/function.sprintf.php

- Calls to *trigger_error*[10] with type `E_USER_DEPRECATED` must be switched to opt-in via `@` operator. Read more at Deprecations;
- Do not use `else`, `elseif`, `break` after `if` and `case` conditions which return or throw something;
- Do not use spaces around `[` offset accessor and before `]` offset accessor;
- Add a `use` statement for every class that is not part of the global namespace;
- When PHPDoc tags like `@param` or `@return` include `null` and other types, always place `null` at the end of the list of types.

Naming Conventions

- Use *camelCase*[11] for PHP variables, function and method names, arguments (e.g. `$acceptableContentTypes`, `hasSession()`);
- Use *snake_case*[12] for configuration parameters and Twig template variables (e.g. `framework.csrf_protection`, `http_status_code`);
- Use namespaces for all PHP classes and *UpperCamelCase*[13] for their names (e.g. `ConsoleLogger`);
- Prefix all abstract classes with `Abstract` except PHPUnit `*TestCase`. Please note some early Symfony classes do not follow this convention and have not been renamed for backward compatibility reasons. However all new abstract classes must follow this naming convention;
- Suffix interfaces with `Interface`;
- Suffix traits with `Trait`;
- Suffix exceptions with `Exception`;
- Use UpperCamelCase for naming PHP files (e.g. `EnvVarProcessor.php`) and snake case for naming Twig templates and web assets (`section_layout.html.twig`, `index.scss`);
- For type-hinting in PHPDocs and casting, use `bool` (instead of `boolean` or `Boolean`), `int` (instead of `integer`), `float` (instead of `double` or `real`);
- Don't forget to look at the more verbose *Conventions* document for more subjective naming considerations.

Service Naming Conventions

- A service name must be the same as the fully qualified class name (FQCN) of its class (e.g. `App\EventSubscriber\UserSubscriber`);
- If there are multiple services for the same class, use the FQCN for the main service and use lowercased and underscored names for the rest of services. Optionally divide them in groups separated with dots (e.g. `something.service_name`, `fos_user.something.service_name`);
- Use lowercase letters for parameter names (except when referring to environment variables with the `%env(VARIABLE_NAME)%` syntax);
- Add class aliases for public services (e.g. alias `Symfony\Component\Something\ClassName` to `something.service_name`).

Documentation

- Add PHPDoc blocks for all classes, methods, and functions (though you may be asked to remove PHPDoc that do not add value);
- Group annotations together so that annotations of the same type immediately follow each other, and annotations of a different type are separated by a single blank line;
- Omit the `@return` tag if the method does not return anything;
- The `@package` and `@subpackage` annotations are not used;
- Don't inline PHPDoc blocks, even when they contain just one tag (e.g. don't put `/** {@inheritdoc} */` in a single line);

10. https://secure.php.net/manual/en/function.trigger-error.php
11. https://en.wikipedia.org/wiki/Camel_case
12. https://en.wikipedia.org/wiki/Snake_case
13. https://en.wikipedia.org/wiki/Camel_case

- When adding a new class or when making significant changes to an existing class, an `@author` tag with personal contact information may be added, or expanded. Please note it is possible to have the personal contact information updated or removed per request to the doc:*core team </contributing/code/core_team>*.

License

- Symfony is released under the MIT license, and the license block has to be present at the top of every PHP file, before the namespace.

Chapter 15
Conventions

The *Coding Standards* document describes the coding standards for the Symfony projects and the internal and third-party bundles. This document describes coding standards and conventions used in the core framework to make it more consistent and predictable. You are encouraged to follow them in your own code, but you don't need to.

Method Names

When an object has a "main" many relation with related "things" (objects, parameters, ...), the method names are normalized:

- `get()`
- `set()`
- `has()`
- `all()`
- `replace()`
- `remove()`
- `clear()`
- `isEmpty()`
- `add()`
- `register()`
- `count()`
- `keys()`

The usage of these methods is only allowed when it is clear that there is a main relation:

- a `CookieJar` has many `Cookie` objects;
- a Service `Container` has many services and many parameters (as services is the main relation, the naming convention is used for this relation);
- a Console `Input` has many arguments and many options. There is no "main" relation, and so the naming convention does not apply.

For many relations where the convention does not apply, the following methods must be used instead (where XXX is the name of the related thing):

Main Relation	Other Relations
get()	getXXX()
set()	setXXX()
n/a	replaceXXX()
has()	hasXXX()
all()	getXXXs()
replace()	setXXXs()
remove()	removeXXX()
clear()	clearXXX()
isEmpty()	isEmptyXXX()
add()	addXXX()
register()	registerXXX()
count()	countXXX()
keys()	n/a

While "setXXX" and "replaceXXX" are very similar, there is one notable difference: "setXXX" may replace, or add new elements to the relation. "replaceXXX", on the other hand, cannot add new elements. If an unrecognized key is passed to "replaceXXX" it must throw an exception.

Deprecations

From time to time, some classes and/or methods are deprecated in the framework; that happens when a feature implementation cannot be changed because of backward compatibility issues, but we still want to propose a "better" alternative. In that case, the old implementation can be **deprecated**.

A feature is marked as deprecated by adding a `@deprecated` phpdoc to relevant classes, methods, properties, ...:

Listing 15-1

```
/**
 * @deprecated since version 2.8, to be removed in 3.0. Use XXX instead.
 */
```

The deprecation message should indicate the version when the class/method was deprecated, the version when it will be removed, and whenever possible, how the feature was replaced.

A PHP `E_USER_DEPRECATED` error must also be triggered to help people with the migration starting one or two minor versions before the version where the feature will be removed (depending on the criticality of the removal):

Listing 15-2

```
@trigger_error('XXX() is deprecated since version 2.8 and will be removed in 3.0. Use XXX instead.',
E_USER_DEPRECATED);
```

Without the *@-silencing operator*[1], users would need to opt-out from deprecation notices. Silencing swaps this behavior and allows users to opt-in when they are ready to cope with them (by adding a custom error handler like the one used by the Web Debug Toolbar or by the PHPUnit bridge).

When deprecating a whole class the `trigger_error()` call should be placed between the namespace and the use declarations, like in this example from *ArrayParserCache*[2]:

1. https://php.net/manual/en/language.operators.errorcontrol.php

Listing 15-3

```
namespace Symfony\Component\ExpressionLanguage\ParserCache;

@trigger_error('The '.__NAMESPACE__.'\ArrayParserCache class is deprecated since version 3.2 and will be
removed in 4.0. Use the Symfony\Component\Cache\Adapter\ArrayAdapter class instead.', E_USER_DEPRECATED);

use Symfony\Component\ExpressionLanguage\ParsedExpression;

/**
 * @author Adrien Brault <adrien.brault@gmail.com>
 *
 * @deprecated ArrayParserCache class is deprecated since version 3.2 and will be removed in 4.0. Use the
Symfony\Component\Cache\Adapter\ArrayAdapter class instead.
 */
class ArrayParserCache implements ParserCacheInterface
```

2. https://github.com/symfony/symfony/blob/3.2/src/Symfony/Component/ExpressionLanguage/ParserCache/ArrayParserCache.php

Chapter 16
Git

This document explains some conventions and specificities in the way we manage the Symfony code with Git.

Pull Requests

Whenever a pull request is merged, all the information contained in the pull request (including comments) is saved in the repository.

You can identify pull request merges as the commit message always follows this pattern:

Listing 16-1
```
merged branch USER_NAME/BRANCH_NAME (PR #1111)
```

The PR reference allows you to have a look at the original pull request on GitHub: *https://github.com/symfony/symfony/pull/1111*. But all the information you can get on GitHub is also available from the repository itself.

The merge commit message contains the original message from the author of the changes. Often, this can help understand what the changes were about and the reasoning behind the changes.

Moreover, the full discussion that might have occurred back then is also stored as a Git note (before March 22 2013, the discussion was part of the main merge commit message). To get access to these notes, add this line to your `.git/config` file:

Listing 16-2
```
fetch = +refs/notes/*:refs/notes/*
```

After a fetch, getting the GitHub discussion for a commit is then a matter of adding `--show-notes=github-comments` to the `git show` command:

Listing 16-3
```
$ git show HEAD --show-notes=github-comments
```

Chapter 17
Symfony Code License

Symfony code is released under *the MIT license*[1]:

Copyright (c) 2004-2019 Fabien Potencier

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Other Symfony Licenses

Check out the *license of the Symfony documentation* and other *Symfony licenses and trademarks*[2].

1. https://en.wikipedia.org/wiki/MIT_License
2. https://symfony.com/license

Chapter 18
Documentation Format

The Symfony documentation uses *reStructuredText*[1] as its markup language and *Sphinx*[2] for generating the documentation in the formats read by the end users, such as HTML and PDF.

reStructuredText

reStructuredText is a plaintext markup syntax similar to Markdown, but much stricter with its syntax. If you are new to reStructuredText, take some time to familiarize with this format by reading the existing *Symfony documentation*[3] source code.

If you want to learn more about this format, check out the *reStructuredText Primer*[4] tutorial and the *reStructuredText Reference*[5].

If you are familiar with Markdown, be careful as things are sometimes very similar but different:

- Lists starts at the beginning of a line (no indentation is allowed);
- Inline code blocks use double-ticks (` ``like this`` `).

Sphinx

Sphinx[6] is a build system that provides tools to create documentation from reStructuredText documents. As such, it adds new directives and interpreted text roles to the standard reST markup. Read more about the *Sphinx Markup Constructs*[7].

1. http://docutils.sourceforge.net/rst.html
2. http://sphinx-doc.org/
3. https://github.com/symfony/symfony-docs
4. http://sphinx-doc.org/rest.html
5. http://docutils.sourceforge.net/docs/user/rst/quickref.html
6. http://sphinx-doc.org/
7. http://sphinx-doc.org/markup/

Syntax Highlighting

PHP is the default syntax highlighter applied to all code blocks. You can change it with the `code-block` directive:

Listing 18-1
```
.. code-block:: yaml

    { foo: bar, bar: { foo: bar, bar: baz } }
```

Besides all of the major programming languages, the syntax highlighter supports all kinds of markup and configuration languages. Check out the list of *supported languages*[8] on the syntax highlighter website.

Configuration Blocks

Whenever you include a configuration sample, use the `configuration-block` directive to show the configuration in all supported configuration formats (`PHP`, `YAML` and `XML`). Example:

Listing 18-2
```
.. configuration-block::

    .. code-block:: yaml

        # Configuration in YAML

    .. code-block:: xml

        <!-- Configuration in XML -->

    .. code-block:: php

        // Configuration in PHP
```

The previous reST snippet renders as follow:

Listing 18-3
```
# Configuration in YAML
```

The current list of supported formats are the following:

Markup Format	Use It to Display
`html`	HTML
`xml`	XML
`php`	PHP
`yaml`	YAML
`twig`	Pure Twig markup
`html+twig`	Twig markup blended with HTML
`html+php`	PHP code blended with HTML
`ini`	INI
`php-annotations`	PHP Annotations

8. http://pygments.org/languages/

Adding Links

The most common type of links are **internal links** to other documentation pages, which use the following syntax:

Listing 18-4
```
:doc:`/absolute/path/to/page`
```

The page name should not include the file extension (`.rst`). For example:

Listing 18-5
```
:doc:`/controller`

:doc:`/components/event_dispatcher`

:doc:`/configuration/environments`
```

The title of the linked page will be automatically used as the text of the link. If you want to modify that title, use this alternative syntax:

Listing 18-6
```
:doc:`Spooling Email </email/spool>`
```

Although they are technically correct, avoid the use of relative internal links such as the following, because they break the references in the generated PDF documentation:

Listing 18-7
```
:doc:`controller`

:doc:`event_dispatcher`

:doc:`environments`
```

Links to the API follow a different syntax, where you must specify the type of the linked resource (`namespace`, `class` or `method`):

Listing 18-8
```
:namespace:`Symfony\\Component\\BrowserKit`

:class:`Symfony\\Component\\Routing\\Matcher\\ApacheUrlMatcher`

:method:`Symfony\\Component\\HttpKernel\\Bundle\\Bundle::build`
```

Links to the PHP documentation follow a pretty similar syntax:

Listing 18-9
```
:phpclass:`SimpleXMLElement`

:phpmethod:`DateTime::createFromFormat`

:phpfunction:`iterator_to_array`
```

New Features, Behavior Changes or Deprecations

If you are documenting a brand new feature, a change or a deprecation that's been made in Symfony, you should precede your description of the change with the corresponding directive and a short description:

For a new feature or a behavior change use the `.. versionadded:: 3.x` directive:

Listing 18-10
```
.. versionadded:: 3.4

    The special ``!`` template prefix was introduced in Symfony 3.4.
```

If you are documenting a behavior change, it may be helpful to *briefly* describe how the behavior has changed:

Listing 18-11
```
.. versionadded:: 3.4

    Support for annotation routing without an external bundle was introduced
    in Symfony 3.4. Prior, you needed to install the SensioFrameworkExtraBundle.
```

For a deprecation use the `.. deprecated:: 3.X` directive:

Listing 18-12
```
.. deprecated:: 3.3

    This technique is discouraged and the ``addClassesToCompile()`` method was
    deprecated in Symfony 3.3 because modern PHP versions make it unnecessary.
```

Whenever a new major version of Symfony is released (e.g. 3.0, 4.0, etc), a new branch of the documentation is created from the `master` branch. At this point, all the `versionadded` and `deprecated` tags for Symfony versions that have a lower major version will be removed. For example, if Symfony 4.0 were released today, 3.0 to 3.4 `versionadded` and `deprecated` tags would be removed from the new `4.0` branch.

Chapter 19

Symfony Documentation License

The Symfony documentation is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License (*CC BY-SA 3.0*[1]).

You are free:

- to *Share* — to copy, distribute and transmit the work;
- to *Remix* — to adapt the work.

Under the following conditions:

- *Attribution* — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work);
- *Share Alike* — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- *Waiver* — Any of the above conditions can be waived if you get permission from the copyright holder;
- *Public Domain* — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license;
- *Other Rights* — In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- *Notice* — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This is a human-readable summary of the *Legal Code (the full license)*[2].

1. http://creativecommons.org/licenses/by-sa/3.0/
2. http://creativecommons.org/licenses/by-sa/3.0/legalcode

Other Symfony Licenses

Check out the *license of the Symfony code* and other *Symfony licenses and trademarks*[3].

3. https://symfony.com/license

Chapter 20

Contributing to the Documentation

Before Your First Contribution

Before contributing, you need to:

- Sign up for a free *GitHub*[1] account, which is the service where the Symfony documentation is hosted.
- Be familiar with the *reStructuredText*[2] markup language, which is used to write Symfony docs. Read *this article* for a quick overview.

Fast Online Contributions

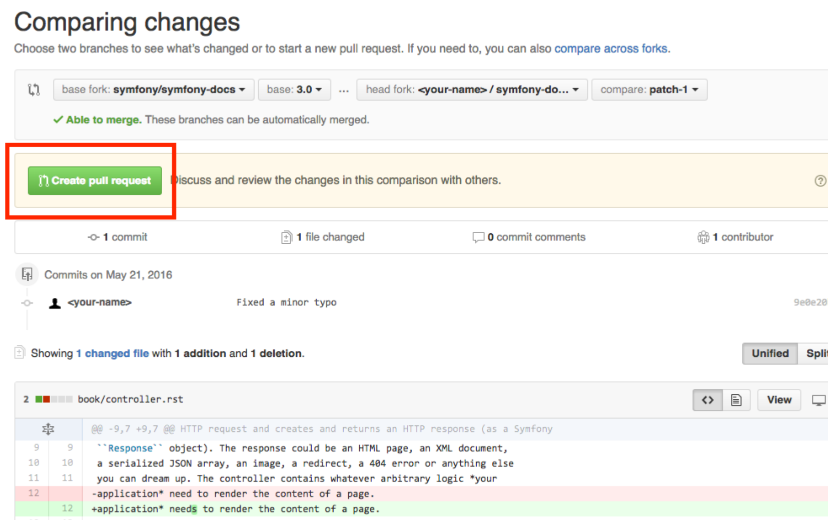
If you're making a relatively small change - like fixing a typo or rewording something - the easiest way to contribute is directly on GitHub! You can do this while you're reading the Symfony documentation.

Step 1. Click on the **edit this page** button on the upper right corner and you'll be redirected to GitHub:

1. `https://github.com/`
2. `http://docutils.sourceforge.net/rst.html`

Step 2. Edit the contents, describe your changes and click on the **Propose file change** button.

Step 3. GitHub will now create a branch and a commit for your changes (forking the repository first if this is your first contribution) and it will also display a preview of your changes:



If everything is correct, click on the **Create pull request** button.

Step 4. GitHub will display a new page where you can do some last-minute changes to your pull request before creating it. For simple contributions, you can safely ignore these options and just click on the **Create pull request** button again.

Congratulations! You just created a pull request to the official Symfony documentation! The community will now review your pull request and (possibly) suggest tweaks.

If your contribution is large or if you prefer to work on your own computer, keep reading this guide to learn an alternative way to send pull requests to the Symfony Documentation.

Your First Documentation Contribution

In this section, you'll learn how to contribute to the Symfony documentation for the first time. The next section will explain the shorter process you'll follow in the future for every contribution after your first one.

Let's imagine that you want to improve the Setup guide. In order to make your changes, follow these steps:

Step 1. Go to the official Symfony documentation repository located at *github.com/symfony/symfony-docs*[3] and click on the **Fork** button to fork the repository to your personal account. This is only needed the first time you contribute to Symfony.

Step 2. Clone the forked repository to your local machine (this example uses the `projects/symfony-docs/` directory to store the documentation; change this value accordingly):

Listing 20-1

```
$ cd projects/
$ git clone git://github.com/YOUR-GITHUB-USERNAME/symfony-docs.git
```

Step 3. Add the original Symfony docs repository as a "Git remote" executing this command:

Listing 20-2

3. `https://github.com/symfony/symfony-docs`

```
1 $ cd symfony-docs/
2 $ git remote add upstream https://github.com/symfony/symfony-docs.git
```

If things went right, you'll see the following when listing the "remotes" of your project:

Listing 20-3
```
1 $ git remote -v
2 origin  git@github.com:YOUR-GITHUB-USERNAME/symfony-docs.git (fetch)
3 origin  git@github.com:YOUR-GITHUB-USERNAME/symfony-docs.git (push)
4 upstream  https://github.com/symfony/symfony-docs.git (fetch)
5 upstream  https://github.com/symfony/symfony-docs.git (push)
```

Fetch all the commits of the upstream branches by executing this command:

Listing 20-4
```
1 $ git fetch upstream
```

The purpose of this step is to allow you work simultaneously on the official Symfony repository and on your own fork. You'll see this in action in a moment.

Step 4. Create a dedicated **new branch** for your changes. Use a short and memorable name for the new branch (if you are fixing a reported issue, use `fix_XXX` as the branch name, where `XXX` is the number of the issue):

Listing 20-5
```
1 $ git checkout -b improve_install_article upstream/3.4
```

In this example, the name of the branch is `improve_install_article` and the `upstream/3.4` value tells Git to create this branch based on the `3.4` branch of the `upstream` remote, which is the original Symfony Docs repository.

Fixes should always be based on the **oldest maintained branch** which contains the error. Nowadays this is the `3.4` branch. If you are instead documenting a new feature, switch to the first Symfony version that included it, e.g. `upstream/3.1`. Not sure? That's ok! Just use the `upstream/master` branch.

Step 5. Now make your changes in the documentation. Add, tweak, reword and even remove any content and do your best to comply with the *Documentation Standards*. Then commit your changes!

Listing 20-6
```
1 # if the modified content existed before
2 $ git add setup.rst
3 $ git commit setup.rst
```

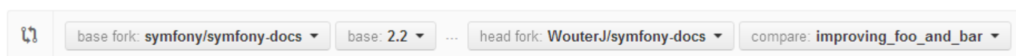
Step 6. Push the changes to your forked repository:

Listing 20-7
```
1 $ git push origin improve_install_article
```

The `origin` value is the name of the Git remote that corresponds to your forked repository and `improve_install_article` is the name of the branch you created previously.

Step 7. Everything is now ready to initiate a **pull request**. Go to your forked repository at `https://github.com/YOUR-GITHUB-USERNAME/symfony-docs` and click on the **Pull Requests** link located in the sidebar.

Then, click on the big **New pull request** button. As GitHub cannot guess the exact changes that you want to propose, select the appropriate branches where changes should be applied:



In this example, the **base fork** should be `symfony/symfony-docs` and the **base** branch should be the `3.4`, which is the branch that you selected to base your changes on. The **head fork** should be your forked

copy of `symfony-docs` and the **compare** branch should be `improve_install_article`, which is the name of the branch you created and where you made your changes.

Step 8. The last step is to prepare the **description** of the pull request. A short phrase or paragraph describing the proposed changes is enough to ensure that your contribution can be reviewed.

Step 9. Now that you've successfully submitted your first contribution to the Symfony documentation, **go and celebrate!** The documentation managers will carefully review your work in short time and they will let you know about any required change.

In case you are asked to add or modify something, don't create a new pull request. Instead, make sure that you are on the correct branch, make your changes and push the new changes:

Listing 20-8

```
$ cd projects/symfony-docs/
$ git checkout improve_install_article

# ... do your changes

$ git push
```

Step 10. After your pull request is eventually accepted and merged in the Symfony documentation, you will be included in the Symfony Documentation Contributors list. Moreover, if you happen to have a *SymfonyConnect*[4] profile, you will get a cool *Symfony Documentation Badge*[5].

Your Next Documentation Contributions

Check you out! You've made your first contribution to the Symfony documentation! Somebody throw a party! Your first contribution took a little extra time because you needed to learn a few standards and setup your computer. But from now on, your contributions will be much easier to complete.

Here is a **checklist** of steps that will guide you through your next contribution to the Symfony docs:

Listing 20-9

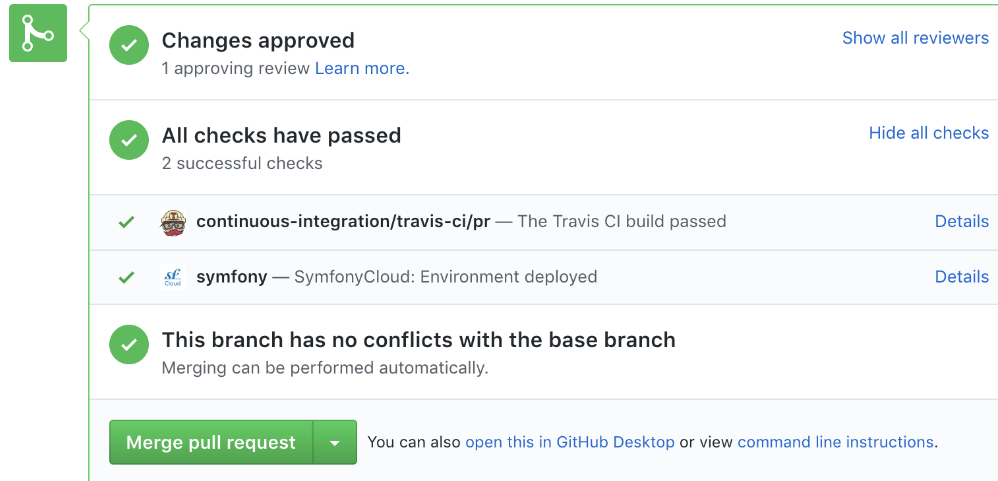
```
# create a new branch based on the oldest maintained version
$ cd projects/symfony-docs/
$ git fetch upstream
$ git checkout -b my_changes upstream/3.4

# ... do your changes

# (optional) add your changes if this is a new content
$ git add xxx.rst

# commit your changes and push them to your fork
$ git commit xxx.rst
$ git push origin my_changes

# ... go to GitHub and create the Pull Request

# (optional) make the changes requested by reviewers and commit them
$ git commit xxx.rst
$ git push
```

After completing your next contributions, also watch your ranking improve on the list of *Symfony Documentation Contributors*[6]. You guessed right: after all this hard work, it's **time to celebrate again!**

4. https://connect.symfony.com/
5. https://connect.symfony.com/badge/36/symfony-documentation-contributor
6. https://symfony.com/contributors/doc

Review your changes

Every GitHub Pull Request is automatically built and deployed by *SymfonyCloud*[7] on a single environment that you can access on your browser to review your changes.



To access the *SymfonyCloud*[8] environment URL, go to your Pull Request page on GitHub, click on the **Show all checks** link and finally, click on the `Details` link displayed for SymfonyCloud service.

Only Pull Requests to maintained branches are automatically built by SymfonyCloud. Check the *roadmap*[9] for maintained branches.

Build the Documentation Locally

If you have Docker installed on your machine, run these commands to build the docs:

Listing 20-10
```
# build the image...
$ docker build . -t symfony-docs

# ...and start the local web server
# (if it's already in use, change the '8080' port by any other port)
$ docker run --rm -p 8080:80 symfony-docs
```

You can now read the docs at `http://127.0.0.1:8080` (if you use a virtual machine, browse its IP instead of localhost; e.g. `http://192.168.99.100:8080`).

If you don't use Docker, follow these steps to build the docs locally:

1. Install *pip*[10] as explained in the *pip installation*[11] article;
2. Install *Sphinx*[12] and *Sphinx Extensions for PHP and Symfony*[13] (depending on your system, you may need to execute this command as root user):

---

7. `https://symfony.com/cloud`
8. `https://symfony.com/cloud`
9. `https://symfony.com/roadmap`
10. `https://pip.pypa.io/en/stable/`
11. `https://pip.pypa.io/en/stable/installing/`
12. `http://sphinx-doc.org/`
13. `https://github.com/fabpot/sphinx-php`

Listing 20-11
```
1  $ pip install sphinx~=1.3.0 git+https://github.com/fabpot/sphinx-php.git
```

3. Run the following command to build the documentation in HTML format:

Listing 20-12
```
1  $ cd _build/
2  $ make html
```

The generated documentation is available in the `_build/html` directory.

Frequently Asked Questions

Why Do my Changes Take so Long to Be Reviewed and/or Merged?

Please be patient. It can take up to several days before your pull request can be fully reviewed. After merging the changes, it could take again several hours before your changes appear on the symfony.com website.

Why Should I Use the Oldest Maintained Branch Instead of the Master Branch?

Consistent with Symfony's source code, the documentation repository is split into multiple branches, corresponding to the different versions of Symfony itself. The `master` branch holds the documentation for the development branch of the code.

Unless you're documenting a feature that was introduced after Symfony 3.4, your changes should always be based on the `3.4` branch. Documentation managers will use the necessary Git-magic to also apply your changes to all the active branches of the documentation.

What If I Want to Submit my Work without Fully Finishing It?

You can do it. But please use one of these two prefixes to let reviewers know about the state of your work:

- `[WIP]` (Work in Progress) is used when you are not yet finished with your pull request, but you would like it to be reviewed. The pull request won't be merged until you say it is ready.
- `[WCM]` (Waiting Code Merge) is used when you're documenting a new feature or change that hasn't been accepted yet into the core code. The pull request will not be merged until it is merged in the core code (or closed if the change is rejected).

Would You Accept a Huge Pull Request with Lots of Changes?

First, make sure that the changes are somewhat related. Otherwise, please create separate pull requests. Anyway, before submitting a huge change, it's probably a good idea to open an issue in the Symfony Documentation repository to ask the managers if they agree with your proposed changes. Otherwise, they could refuse your proposal after you put all that hard work into making the changes. We definitely don't want you to waste your time!

Chapter 21

Documentation Standards

Contributions must follow these standards to match the style and tone of the rest of the Symfony documentation.

Sphinx

- The following characters are chosen for different heading levels: level 1 is `=` (equal sign), level 2 `-` (dash), level 3 `~` (tilde), level 4 `.` (dot) and level 5 `"` (double quote);
- Each line should break approximately after the first word that crosses the 72nd character (so most lines end up being 72-78 characters);
- The `::` shorthand is *preferred* over `.. code-block:: php` to begin a PHP code block unless it results in the marker being on its own line (read *the Sphinx documentation*[1] to see when you should use the shorthand);
- Inline hyperlinks are **not** used. Separate the link and their target definition, which you add on the bottom of the page;
- Inline markup should be closed on the same line as the open-string;

Example

Listing 21-1

```
Example
=======

When you are working on the docs, you should follow the
`Symfony Documentation`_ standards.

Level 2
-------

A PHP example would be::

    echo 'Hello World';

Level 3
```

1. `http://sphinx-doc.org/rest.html#source-code`

```
15 ~~~~~~~
16
17 .. code-block:: php
18
19     echo 'You cannot use the :: shortcut here';
20
21 .. _`Symfony Documentation`: https://symfony.com/doc
```

Code Examples

- The code follows the *Symfony Coding Standards* as well as the *Twig Coding Standards*[2];
- The code examples should look real for a web application context. Avoid abstract or trivial examples (`foo`, `bar`, `demo`, etc.);
- The code should follow the *Symfony Best Practices*.
- Use `Acme` when the code requires a vendor name;
- Use `example.com` as the domain of sample URLs and `example.org` and `example.net` when additional domains are required. All of these domains are *reserved by the IANA*[3].
- To avoid horizontal scrolling on code blocks, we prefer to break a line correctly if it crosses the 85th character;
- When you fold one or more lines of code, place `...` in a comment at the point of the fold. These comments are: `// ...` (php), `# ...` (yaml/bash), `{# ... #}` (twig), `<!-- ... -->` (xml/html), `; ...` (ini), `...` (text);
- When you fold a part of a line, e.g. a variable value, put `...` (without comment) at the place of the fold;
- Description of the folded code: (optional)
  - If you fold several lines: the description of the fold can be placed after the `...`;
  - If you fold only part of a line: the description can be placed before the line;
- If useful to the reader, a PHP code example should start with the namespace declaration;
- When referencing classes, be sure to show the `use` statements at the top of your code block. You don't need to show *all* `use` statements in every example, just show what is actually being used in the code block;
- If useful, a `codeblock` should begin with a comment containing the filename of the file in the code block. Don't place a blank line after this comment, unless the next line is also a comment;
- You should put a `$` in front of every bash line.

Formats

Configuration examples should show all supported formats using configuration blocks. The supported formats (and their orders) are:

- **Configuration** (including services): YAML, XML, PHP
- **Routing**: Annotations, YAML, XML, PHP
- **Validation**: Annotations, YAML, XML, PHP
- **Doctrine Mapping**: Annotations, YAML, XML, PHP
- **Translation**: XML, YAML, PHP

Example

Listing 21-2

2. https://twig.symfony.com/doc/2.x/coding_standards.html
3. http://tools.ietf.org/html/rfc2606#section-3

```
// src/Foo/Bar.php
namespace Foo;

use Acme\Demo\Cat;
// ...

class Bar
{
    // ...

    public function foo($bar)
    {
        // set foo with a value of bar
        $foo = ...;

        $cat = new Cat($foo);

        // ... check if $bar has the correct value

        return $cat->baz($bar, ...);
    }
}
```

In YAML you should put a space after `{` and before `}` (e.g. `{ _controller: ... }`), but this should not be done in Twig (e.g. `{'hello' : 'value'}`).

Files and Directories

- When referencing directories, always add a trailing slash to avoid confusions with regular files (e.g. "execute the `console` script located at the `bin/` directory").
- When referencing file extensions explicitly, you should include a leading dot for every extension (e.g. "XML files use the `.xml` extension").
- When you list a Symfony file/directory hierarchy, use `your-project/` as the top level directory. E.g.

Listing 21-3

```
your-project/
├─ app/
├─ src/
├─ vendor/
└─ ...
```

English Language Standards

Symfony documentation uses the United States English dialect, commonly called *American English*[4]. The *American English Oxford Dictionary*[5] is used as the vocabulary reference.

In addition, documentation follows these rules:

- **Section titles**: use a variant of the title case, where the first word is always capitalized and all other words are capitalized, except for the closed-class words (read Wikipedia article about *headings and titles*[6]).

4. https://en.wikipedia.org/wiki/American_English
5. http://en.oxforddictionaries.com/definition/american_english/
6. https://en.wikipedia.org/wiki/Letter_case#Headings_and_publication_titles

E.g.: The Vitamins are in my Fresh California Raisins

- **Punctuation**: avoid the use of *Serial (Oxford) Commas*[7];
- **Pronouns**: avoid the use of *nosism*[8] and always use *you* instead of *we*. (i.e. avoid the first person point of view: use the second instead);
- **Gender-neutral language**: when referencing a hypothetical person, such as *"a user with a session cookie"*, use gender-neutral pronouns (they/their/them). For example, instead of:
    - he or she, use they
    - him or her, use them
    - his or her, use their
    - his or hers, use theirs
    - himself or herself, use themselves

7. https://en.wikipedia.org/wiki/Serial_comma
8. https://en.wikipedia.org/wiki/Nosism

Chapter 22

Translations

The official Symfony documentation is published only in English. You can read about the reasons in *this blog post*[1].

We have taken steps to improve the experience when using *Google Translate*[2] to prevent code blocks from being translated.

To translate any page in our documentation please copy any URL from the documentation and paste it into the form on the Google Translate site.

1. https://symfony.com/blog/discontinuing-the-symfony-community-translations
2. https://translate.google.com

Chapter 23

The Release Process

This document explains the process followed by the Symfony project to develop, release and maintain its different versions.

Symfony releases follow the *semantic versioning*[1] strategy and they are published through a *time-based model*:

- A new **Symfony patch version** (e.g. 2.8.15, 4.1.7) comes out roughly every month. It only contains bug fixes, so you can safely upgrade your applications;
- A new **Symfony minor version** (e.g. 2.8, 3.2, 4.1) comes out every *six months*: one in *May* and one in *November*. It contains bug fixes and new features, but it doesn't include any breaking change, so you can safely upgrade your applications;
- A new **Symfony major version** (e.g. 3.0, 4.0) comes out every *two years*. It can contain breaking changes, so you may need to do some changes in your applications before upgrading.

Subscribe to Symfony Roadmap notifications[2] to receive an email when a new Symfony version is published or when a Symfony version reaches its end of life.

Development

The full development period for any major or minor version lasts six months and is divided into two phases:

- **Development**: *Four months* to add new features and to enhance existing ones;
- **Stabilization**: *Two months* to fix bugs, prepare the release, and wait for the whole Symfony ecosystem (third-party libraries, bundles, and projects using Symfony) to catch up.

During the development phase, any new feature can be reverted if it won't be finished in time or if it won't be stable enough to be included in the current final release.

1. https://semver.org/
2. https://symfony.com/account

Check out the *Symfony Roadmap*[3] to learn more about any specific version.

Maintenance

Starting from the Symfony 3.x branch, the number of minor versions is limited to five per branch (X.0, X.1, X.2, X.3 and X.4). The last minor version of a branch (e.g. 3.4, 4.4, 5.4) is considered a **long-term support version** and the other ones are considered **standard versions**:

Version Type	Bugs are fixed for...	Security issues are fixed for...
Standard	8 months	14 months
Long-Term Support (LTS)	3 years	4 years

After the active maintenance of a Symfony version has ended, you can get *professional Symfony support*[4] from SensioLabs, the company which sponsors the Symfony project.

Backward Compatibility

Our *Backward Compatibility Promise* is very strict and allows developers to upgrade with confidence from one minor version of Symfony to the next one.

When a feature implementation cannot be replaced with a better one without breaking backward compatibility, Symfony deprecates the old implementation and adds a new preferred one along side. Read the conventions document to learn more about how deprecations are handled in Symfony.

Rationale

This release process was adopted to give more *predictability* and *transparency*. It was discussed based on the following goals:

- Shorten the release cycle (allow developers to benefit from the new features faster);
- Give more visibility to the developers using the framework and Open-Source projects using Symfony;
- Improve the experience of Symfony core contributors: everyone knows when a feature might be available in Symfony;
- Coordinate the Symfony timeline with popular PHP projects that work well with Symfony and with projects using Symfony;
- Give time to the Symfony ecosystem to catch up with the new versions (bundle authors, documentation writers, translators, ...);
- Give companies a strict and predictable timeline they can rely on to plan their own projects development.

The six month period was chosen as two releases fit in a year. It also allows for plenty of time to work on new features and it allows for non-ready features to be postponed to the next version without having to wait too long for the next cycle.

3. `https://symfony.com/roadmap#checker`
4. `https://sensiolabs.com/`

The dual maintenance mode was adopted to make every Symfony user happy. Fast movers, who want to work with the latest and the greatest, use the standard version: a new version is published every six months, and there is a two months period to upgrade. Companies wanting more stability use the LTS versions: a new version is published every two years and there is a year to upgrade.

Chapter 24

Respectful Review Comments

Reviewing issues and pull requests is a great way to get started with contributing to the Symfony community. Anyone can do it! But before you give a comment, take a step back and think, is what you are about to say actually what you intend?

Communicating over the Internet with nothing but text can pose a big challenge, especially if you remember that the Symfony community is world-wide and is composed of a wide variety of people with differing ideas and opinions.

Not everyone speaks English or is able to use a keyboard. Some might have dyslexia or similar conditions that affect their writing.

Not to mention that some might have a bad experience from previous contributions (to other projects).

You're not alone in this. This guide will try to help you write constructive, respectful and helpful reviews and replies.

This guide is not about lecturing you to "conform" or give-up your ideas and opinions but helping you to better communicate, prevent possible confusion, and keeping the Symfony community a welcoming place for everyone. **You are free to disagree with someone's opinions, but don't be disrespectful.**

First of, accept that many programming decisions are opinions. Discuss trade offs, which you prefer, and reach a resolution quickly. It's not about being right or wrong, but using what works.

Tone of Voice

We don't expect you to be completely formal, or to even write error-free English. Just remember this: don't swear, and be respectful to others.

Don't reply in anger or with an aggressive tone. If you're angry, we understand that, but swearing/cursing and name calling doesn't really encourage anyone to help you. Take a deep breath, count to 10 and try to *clearly* explain what problems you encounter.

Inclusive Language

In an effort to be inclusive to a wide group of people, it's recommended to use personal pronouns that don't suggest a particular gender. Unless someone has stated their pronouns, use "they", "them" instead of "he", "she", "his", "hers", "his/hers", "he/she", etc.

Try to avoid using wording that may be considered excluding, needlessly gendered (e.g. words that have a male or female base), racially motivated or singles out a particular group in society. For example, it's recommended to use words like "folks", "team", "everyone" instead of "guys", "ladies", "yanks", etc.

Giving Positive Feedback

While reviewing issues and pull requests you may run into some suggestions (including patches) that don't reflect your ideas, are not good, or downright wrong.

Now, when you prepare your comment, consider the amount of work and time the author has spent on their idea and how your response would make them feel.

Did you correctly understand their intention? Or are you making assumptions? Whatever your response, be explicit. Remember people don't always understand your intentions online.

Avoid using terms that could be seen as referring to personal traits ("dumb", "stupid"). Assume everyone is intelligent and well-meaning.

Good questions avoid judgement and avoid assumptions about the author's perspective.

Maybe you can ask for clarification? Suggest an alternative? Or provide a simple explanation *why* you disagree with their proposal.

- `This looks wrong. Are you sure it's correct?` (eg. typo/syntax error)
- `What do you think of "RequestFactory" instead of RequestCreator?`

Even if something *is* really wrong or "a bad idea", stay respectful and don't get into endless you-are-wrong discussions or "flame wars".

Don't use hyperbole ("always", "never", "endlessly", "nothing", "worst", "horrible", "terrible").

Don't: *"I don't like how you wrote this code"* - there is no clear explanation why you don't like how it's written.

Better: *"I find it hard to read this code as there are many nested if statements, can you make it more readable? By encapsulating some of the details or maybe adding some comments to explain the overall logic."* - You explain why you find the code hard to read *and* give some suggestions for improvement.

If a piece of code is in fact wrong, explain why:

- "This code doesn't comply with Symfony's CS rules. Please see [...] for details."
- "Symfony 3 still uses PHP 5 and doesn't allow the usage of scalar type-hints."
- "I think the code is less readable now." - careful here, be sure explain why you think the code is less readable, and maybe give some suggestions?

Examples of valid reasons to reject:

- "We tried that in the past (link to the relevant PR) but we needed to revert it for XXX reason."
- "That change would introduce too many merge conflicts when merging up Symfony branches. In the past we've always rejected changes like this."
- "I profiled this change and it hurts performance significantly" - if you don't profile, it's an opinion, so we can ignore
- "Code doesn't match Symfony's CS rules (e.g. use `[]` instead of `array()`)"

- "We only provide integration with very popular projects (e.g. we integrate Bootstrap but not your own CSS framework)"
- "This would require adding lots of code and making lots of changes for a feature that doesn't look so important. That could hurt maintaining in the future."

Asking for Changes

Rarely something is perfect from the start, while the code itself is good. It may not be optimal or conform the Symfony coding style.

Again, understand the author already spent time on the issue and asking for (small) changes may be misinterpreted or seen as a personal attack.

Be thankful for their work (so far), stay positive and really help them to make the contribution a great one. *Especially if they are a first time contributor*.

Use words like "Please", "Thank you" and "Could you" instead of making demands;

- "Thank you for your work so far. I left some suggestions for improvement to make the code more readable."
- "Your code contains some coding-style problems, can you fix these before we merge? Thank you"
- "Please use 4 spaces instead of tabs", "This needs be on the previous line";

During a pull request review you can usually leave more then one comment, you don't have to use "Please" all the time. But it wouldn't hurt.

It may not seem like much, but saying "Thank you" does make others feel more welcome.

Preventing Escalations

Sometimes when people receive feedback they may get defensive. In that case, it is better to try to approach the discussion in a different way, to not escalate further.

If you want someone to mediate, please join the #contribs channel on *Symfony Slack*[1], to have a safe environment and keep working together on the common goals.

Using Humor

In short: Extreme misbehavior will not be tolerated and may even get you banned; Keep it real and friendly.

Don't use sarcasm for a serious topic, that's not something that belongs to the Symfony community. And don't marginalize someone's problems; `Well I guess that's not supposed to happen? ?`.

Even if someone's explanation is "inviting to joke about it", it's a real problem to them. Making jokes about this doesn't help with solving their problem and only makes them *feel stupid*. Instead try to discover what the problem is really about.

1. `https://symfony.com/slack-invite`

Final Words

Don't feel bad if you "failed" to follow these tips. As long as your intentions were good and you didn't really offend or insult anyone; you can explain you misunderstood, you didn't mean to marginalize or simply failed.

But don't say it "just because", if your apology is not really meant you *will* lose credibility and respect from other developers.

Do unto others as you would have them do unto you.

Chapter 25
Community Reviews

Symfony is an open-source project driven by a large community. If you don't feel ready to contribute code or patches, reviewing issues and pull requests (PRs) can be a great start to get involved and give back. In fact, people who "triage" issues are the backbone to Symfony's success!

Communicating in a way where your words come across as intended can be difficult. Please read through the *Respectful Review Comments* guidelines.

Why Reviewing Is Important

Community reviews are essential for the development of the Symfony framework, since there are many more pull requests and bug reports than there are members in the Symfony core team to review, fix and merge them.

On the *Symfony issue tracker*[1], you can find many items in a *Needs Review*[2] status:

- **Bug Reports**: Bug reports need to be checked for completeness. Is any important information missing? Can the bug be *easily* reproduced?
- **Pull Requests**: Pull requests contain code that fixes a bug or implements new functionality. Reviews of pull requests ensure that they are implemented properly, are covered by test cases, don't introduce new bugs and maintain backward compatibility.

Note that **anyone who has some basic familiarity with Symfony and PHP can review bug reports and pull requests**. You don't need to be an expert to help.

1. https://github.com/symfony/symfony/issues
2. https://github.com/symfony/symfony/labels/Status%3A%20Needs%20Review

Be Constructive

Before you begin, remember that you are looking at the result of someone else's hard work. A good review comment thanks the contributor for their work, identifies what was done well, identifies what should be improved and suggests a next step.

Create a GitHub Account

Symfony uses *GitHub*[3] to manage bug reports and pull requests. If you want to do reviews, you need to *create a GitHub account*[4] and log in.

The Bug Report Review Process

A good way to get started with reviewing is to pick a bug report from the *bug reports in need of review*[5].

The steps for the review are:

1. **Is the Report Complete?**

   Good bug reports contain a link to a fork of the *Symfony Standard Edition*[6] (the "reproduction project") that reproduces the bug. If it doesn't, the report should at least contain enough information and code samples to reproduce the bug.

2. **Reproduce the Bug**

   Download the reproduction project and test whether the bug can be reproduced on your system. If the reporter did not provide a reproduction project, create one by *forking*[7] the *Symfony Standard Edition*[8].

3. **Update the Issue Status**

   At last, add a comment to the bug report. **Thank the reporter for reporting the bug**. Include the line `Status: <status>` in your comment to trigger our *Carson Bot*[9] which updates the status label of the issue. You can set the status to one of the following:

   Needs Work If the bug *does not* contain enough information to be reproduced, explain what information is missing and move the report to this status.

   Works for me If the bug *does* contain enough information to be reproduced but works on your system, or if the reported bug is a feature and not a bug, provide a short explanation and move the report to this status.

   Reviewed If you can reproduce the bug, move the report to this status. If you created a reproduction project, include the link to the project in your comment.

Example

Here is a sample comment for a bug report that could be reproduced:

Listing 25-1

```
Thank you @weaverryan for creating this bug report! This indeed looks
like a bug. I reproduced the bug in the "kernel-bug" branch of
https://github.com/webmozart/symfony-standard.
```

3. https://github.com
4. https://help.github.com/articles/signing-up-for-a-new-github-account/
5. https://github.com/symfony/symfony/issues?utf8=%E2%9C%93&q=is%3Aopen+is%3Aissue+label%3A%22Bug%22+label%3A%22Status%3A+Needs+Review%22+
6. https://github.com/symfony/symfony-standard
7. https://help.github.com/articles/fork-a-repo/
8. https://github.com/symfony/symfony-standard
9. https://github.com/carsonbot/carsonbot

```
4
5 Status: Reviewed
```

The Pull Request Review Process

The process for reviewing pull requests (PRs) is similar to the one for bug reports. Reviews of pull requests usually take a little longer since you need to understand the functionality that has been fixed or added and find out whether the implementation is complete.

It is okay to do partial reviews! If you do a partial review, comment how far you got and leave the PR in "Needs Review" state.

Pick a pull request from the *PRs in need of review*[10] and follow these steps:

1. **Is the PR Complete**?

   Every pull request must contain a header that gives some basic information about the PR. You can find the template for that header in the Contribution Guidelines.

2. **Is the Base Branch Correct?**

   GitHub displays the branch that a PR is based on below the title of the pull request. Is that branch correct?

   - Bugs should be fixed in the oldest, maintained version that contains the bug. Check *Symfony's Release Schedule* to find the oldest currently supported version.
   - New features should always be added to the current development version. Check the *Symfony Roadmap*[11] to find the current development version.

3. **Reproduce the Problem**

   Read the issue that the pull request is supposed to fix. Reproduce the problem on a clean *Symfony Standard Edition*[12] project and try to understand why it exists. If the linked issue already contains such a project, install it and run it on your system.

4. **Review the Code**

   Read the code of the pull request and check it against some common criteria:

   - Does the code address the issue the PR is intended to fix/implement?
   - Does the PR stay within scope to address *only* that issue?
   - Does the PR contain automated tests? Do those tests cover all relevant edge cases?
   - Does the PR contain sufficient comments to easily understand its code?
   - Does the code break backward compatibility? If yes, does the PR header say so?
   - Does the PR contain deprecations? If yes, does the PR header say so? Does the code contain `trigger_error()` statements for all deprecated features?
   - Are all deprecations and backward compatibility breaks documented in the latest UPGRADE-X.X.md file? Do those explanations contain "Before"/"After" examples with clear upgrade instructions?

   Eventually, some of these aspects will be checked automatically.

5. **Test the Code**

10. https://github.com/symfony/symfony/issues?utf8=%E2%9C%93&q=is%3Aopen+is%3Apr+label%3A%22Status%3A+Needs+Review%22+
11. https://symfony.com/roadmap
12. https://github.com/symfony/symfony-standard

Take your project from step 3 and test whether the PR works properly. Replace the Symfony project in the `vendor` directory by the code in the PR by running the following Git commands. Insert the PR ID (that's the number after the `#` in the PR title) for the `<ID>` placeholders:

Listing 25-2
```
1 $ cd vendor/symfony/symfony
2 $ git fetch origin pull/<ID>/head:pr<ID>
3 $ git checkout pr<ID>
```

For example:

Listing 25-3
```
1 $ git fetch origin pull/15723/head:pr15723
2 $ git checkout pr15723
```

Now you can *test the project* against the code in the PR.

6. **Update the PR Status**

   At last, add a comment to the PR. **Thank the contributor for working on the PR**. Include the line `Status: <status>` in your comment to trigger our *Carson Bot*[13] which updates the status label of the issue. You can set the status to one of the following:

   Needs Work If the PR is not yet ready to be merged, explain the issues that you found and move it to this status.

   Reviewed If the PR satisfies all the checks above, move it to this status. A core contributor will soon look at the PR and decide whether it can be merged or needs further work.

Example

Here is a sample comment for a PR that is not yet ready for merge:

Listing 25-4
```
1 Thank you @weaverryan for working on this! It seems that your test
2 cases don't cover the cases when the counter is zero or smaller.
3 Could you please add some tests for that?
4
5 Status: Needs Work
```

13. `https://github.com/carsonbot/carsonbot`

Chapter 26
Mentoring

Reading the *contributing* is already a great way to get started on becoming a Symfony contributor. However, sometimes it might still seem overwhelming - contributing can be complex! For this purpose we created a dedicated *Symfony Slack*[1] channel called *#mentoring*[2] to connect new contributors to long-time contributors. This is a great way to get one-on-one advice on the entire process. These long-time contributors do really want to help new contributors - so feel free to ask anything!

1. https://symfony.com/slack-invite
2. https://symfony-devs.slack.com/messages/mentoring

Chapter 27
Speaker Mentoring

The Symfony community benefits greatly when as many people as possible share their knowledge and experience with others. Every different point of view adds to our collective understanding of how to best use and evolve the code, design patterns and architecture provided within the Symfony community. Because of this, we specifically want to hear from long-time contributors and new users, who often come across entirely different challenges with a totally fresh new look and perspective.

How to get started

Giving a first talk at a conference can seem quite intimidating. But don't worry! At one time, every speaker went through the same process. And so, we want to make sure that as many people as possible are empowered to take this path if they are motivated. We have collected a few resources with advice to get started. More importantly, we can connect experienced speakers with people who are just taking their first steps in this area:

A good first step might be to give a talk at a local user group to a smaller crowd that one knows more intimately. A next step could be to give a talk at conference in your first language.

The best way to find people that can review your talk idea or slides is the *#speaker-mentoring*[1] channel on *Symfony Slack*[2]. There are many seasoned speakers with knowledge in various parts of Symfony that are motivated to help you get started on your path towards becoming a public speaker. They can even do practice runs via video chat! Furthermore, they can also be an ally when it comes to the day of giving the talk at a conference!

A great resource with advice on everything related to *public speaking*[3] is a collection of links maintained by VM (Vicky) Brasseur. It covers everything from finding a conference call for proposals, how to refine a proposal, to how to put together slide decks to practical tips for preparation and talk delivery.

1. `https://symfony-devs.slack.com/messages/speaker-mentoring`
2. `https://symfony.com/slack-invite`
3. `https://github.com/vmbrasseur/Public_Speaking`

Chapter 28

Other Resources

In order to follow what is happening in the community you might find helpful these additional resources:

- List of open *pull requests*[1]
- List of recent *commits*[2]
- List of open *bugs and enhancements*[3]
- List of open source *bundles*[4]

1. https://github.com/symfony/symfony/pulls
2. https://github.com/symfony/symfony/commits/master
3. https://github.com/symfony/symfony/issues
4. https://github.com/search?q=topic%3Asymfony-bundle&type=Repositories

Chapter 29
Diversity Initiative Governance

Membership

Membership of Symfony's Diversity Initiative is open to any member of the Symfony community; to avoid the risk of elitism or meritocracy, no requirement is needed to be involved. All members, at any time, are invited to put forward ideas and suggestions as a proposal for an actionable item.

Guidance

The project leader, Fabien Potencier, is responsible for publicly appointing five (5) members of the initiative to provide guidance and drive it forward, but also retains the right to revoke any of the appointed members at any time. This guidance team should:

- Be committed to the initiative's cause and have joined because they want to help the initiative to deliver its purpose most effectively for the community's benefit.
- Recognize that meeting the initiative's purpose is an ongoing effort.
- Be committed to good governance and want to contribute to the initiative's continued improvement.

The current guidance team is composed of the following people (in alphabetical order):

- **Lukas Kahwe Smith** (*lsmith77*[1]);
- **Michelle Sanver** (*michellesanver*[2]);
- **Nicolas Grekas** (*nicolas-grekas*[3]);
- **Timo Bakx** (*TimoBakx*[4]);
- **Zan Baldwin** (*zanbaldwin*[5]).

1. https://github.com/lsmith77/
2. https://github.com/michellesanver/
3. https://github.com/nicolas-grekas/
4. https://github.com/TimoBakx/
5. https://github.com/zanbaldwin/

Veto

The project leader (Fabien Potencier) will have the right to veto any actionable item, regardless of the vote of the initiative's guidance team. The project leader may, at their discretion, also appoint other people from among the initiative's guidance team to also have the right to veto - in such a case these people are expected to use appropriate judgement to know when to use a "no" vote or a veto. Any single veto will reject an actionable item.

The purpose of having members with the right to veto is to prevent a "people's majority" from overruling the core interests of the Symfony project. This will encourage communication between proposing members, the initiative's guidance team and the Core Team to create realistic proposals, and in return any veto will come with a full explanation (not just a justification).

Advice Process

When a proposal on an actionable item is ready to be decided on, insight from the community (advice, general consensus, or non-binding poll) should be requested from the wider community - this will aim to include both those who will be meaningfully affected and those with meaningful expertise in the matter at hand. This feedback will enable the guidance team to have the confidence to vote for the best possible decision according to the information they have available, knowing that the responsibility they accept for said vote is justified.

Voting

The guidance team have the right to vote on proposals for actionable items. The quorum of "yes" or "no" votes required for a decision to be considered valid is at least 75% of active, appointed members of the guidance team - to abstain from voting means that vote will not be counted towards the quorum. For an actionable item to pass, approval from greater than 50% of the voting guidance team members is required. Use or management of finances/donations require at least a two-thirds majority to pass.

For transparency and ease-of-understanding, this means only the following combinations of votes will result in an actionable item passing:

For	Against	Abstain
5	0	0
4	1	0
3	2	0
4	0	1
3	1	1

Guidance Principles

Purpose

The initiative should be led by an effective guidance team that provides strategic guidance in line with the initiative's aims and values, including a shared understanding with fellow initiative members to ensure that these are being delivered effectively and sustainably.

Integrity

The guidance team should act with integrity: adopting values which help achieve the initiative's purposes, even where difficult or unpopular decisions are required. Guidance team members should undertake their duties, aware of the importance of confidence and trust in the initiative from the wider community, and ultimately acknowledges shared responsibility for the reputation of the Symfony project like the Core Team.

Decision-making Effectiveness

Guidance members should work as an effective team, using the appropriate balance of skills, experience, backgrounds and knowledge to make sure its decision-making processes are informed and equitable. Risk assessment and management systems should be set up and monitored.

Openness and Accountability

The behavior and conduct of the initiative's guidance team sets the tone for the rest of the community. The guidance team should lead by example to create a culture that enables members to feel it is safe to suggest, question and challenge - rather than avoid - difficult ideas and topics. The team should guide the initiative in being transparent, accountable and open.

Adaptability

The initiative should establish processes that do not require any one person to hold specific positions while being adaptable to accommodate unforeseen needs of the community, especially as membership and involvement grows over time (changes to guidance team member appointment will have to be approved by the current system, which is Fabien Potencier).