[![master](https://travis-ci.org/checkpoint-restore/criu.svg?branch=master)](https://travis-ci.org/checkpoint-restore/criu)

[![development](https://travis-ci.org/checkpoint-restore/criu.svg?branch=criu-dev)](https://travis-ci.org/checkpoint-restore/criu)

[![Codacy Badge](https://api.codacy.com/project/badge/Grade/55251ec7db28421da4481fc7c1cb0cee)](https://www.codacy.com/app/xemul/criu?utm\_source=github.com&amp;utm\_medium=referral&amp;utm\_content=xemul/criu&amp;utm\_campaign=Badge\_Grade)

<p align="center"><img src="https://criu.org/w/images/1/1c/CRIU.svg" width="128px"/></p>

## How to contribute to CRIU

CRIU project is (almost) the never-ending story, because we have to always keep up with the

Linux kernel supporting checkpoint and restore for all the features it provides. Thus we're

looking for contributors of all kinds -- feedback, bug reports, testing, coding, writing, etc.

Here are some useful hints to get involved.

\* We have both -- [very simple](https://github.com/checkpoint-restore/criu/issues?q=is%3Aissue+is%3Aopen+label%3Aenhancement) and [more sophisticated](https://github.com/checkpoint-restore/criu/issues?q=is%3Aissue+is%3Aopen+label%3A%22new+feature%22) coding tasks;

\* CRIU does need [extensive testing](https://github.com/checkpoint-restore/criu/issues?q=is%3Aissue+is%3Aopen+label%3Atesting);

\* Documentation is always hard, we have [some information](https://criu.org/Category:Empty\_articles) that is to be extracted from people's heads into wiki pages as well as [some texts](https://criu.org/Category:Editor\_help\_needed) that all need to be converted into useful articles;

\* Feedback is expected on the GitHub issues page and on the [mailing list](https://lists.openvz.org/mailman/listinfo/criu);

\* We accept GitHub pull requests and this is the preferred way to contribute to CRIU. If you prefer to send patches by email, you are welcome to send them to [CRIU development mailing list](https://lists.openvz.org/mailman/listinfo/criu).

Below we describe in more detail recommend practices for CRIU development.

\* Spread the word about CRIU in [social networks](http://criu.org/Contacts);

\* If you're giving a talk about CRIU -- let us know, we'll mention it on the [wiki main page](https://criu.org/News/events);

### Setting up the development environment

Although `criu` could be run as non-root (see [Security](https://criu.org/Security)), development is better to be done as root. For example, some tests require root. So, it would be a good idea to set up some recent Linux distro on a virtual machine.

### Get the source code

The CRIU sources are tracked by Git. Official CRIU repo is at https://github.com/checkpoint-restore/criu.

The repository may contain multiple branches. Development happens in the \*\*criu-dev\*\* branch.

To clone CRIU repo and switch to the proper branch, run:

```

git clone https://github.com/checkpoint-restore/criu criu

cd criu

git checkout criu-dev

```

### Compile

First, you need to install compile-time dependencies. Check [Installation dependencies](https://criu.org/Installation#Dependencies) for more info.

To compile CRIU, run:

```

make

```

This should create the `./criu/criu` executable.

## Edit the source code

If you use ctags, you can generate the ctags file by running

```

make tags

```

When you change the source code, please keep in mind the following code conventions:

\* we prefer tabs and indentations to be 8 characters width

\* CRIU mostly follows [Linux kernel coding style](https://www.kernel.org/doc/Documentation/process/coding-style.rst), but we are less strict than the kernel community.

Other conventions can be learned from the source code itself. In short, make sure your new code

looks similar to what is already there.

## Test your changes

CRIU comes with an extensive test suite. To check whether your changes introduce any regressions, run

```

make test

```

The command runs [ZDTM Test Suite](https://criu.org/ZDTM\_Test\_Suite). Check for any error messages produced by it.

In case you'd rather have someone else run the tests, you can use travis-ci for your

own GitHub fork of CRIU. It will check the compilation for various supported platforms,

as well as run most of the tests from the suite. See https://travis-ci.org/checkpoint-restore/criu

for more details.

## Describe your changes

Describe your problem. Whether your change is a one-line bug fix or

5000 lines of a new feature, there must be an underlying problem that

motivated you to do this work. Convince the reviewer that there is a

problem worth fixing and that it makes sense for them to read past the

first paragraph.

Once the problem is established, describe what you are actually doing

about it in technical detail. It's important to describe the change

in plain English for the reviewer to verify that the code is behaving

as you intend it to.

Solve only one problem per commit. If your description starts to get

long, that's a sign that you probably need to split up your commit.

See [Separate your changes](#separate-your-changes).

Describe your changes in imperative mood, e.g. "make xyzzy do frotz"

instead of "[This commit] makes xyzzy do frotz" or "[I] changed xyzzy

to do frotz", as if you are giving orders to the codebase to change

its behaviour.

If your change fixes a bug in a specific commit, e.g. you found an issue using

`git bisect`, please use the `Fixes:` tag with the abbreviation of

the SHA-1 ID, and the one line summary. For example:

```

Fixes: 9433b7b9db3e ("make: use cflags/ldflags for config.h detection mechanism")

```

The following `git config` settings can be used to add a pretty format for

outputting the above style in the `git log` or `git show` commands:

```

[pretty]

fixes = Fixes: %h (\"%s\")

```

If your change address an issue listed in GitHub, please use `Fixes:` tag with the number of the issue. For instance:

```

Fixes: #339

```

You may refer to [How to Write a Git Commit

Message](https://chris.beams.io/posts/git-commit/) article for

recommendations for good commit message.

## Separate your changes

Separate each \*\*logical change\*\* into a separate commit.

For example, if your changes include both bug fixes and performance

enhancements for a single driver, separate those changes into two

or more commits. If your changes include an API update, and a new

driver which uses that new API, separate those into two commits.

On the other hand, if you make a single change to numerous files,

group those changes into a single commit. Thus a single logical change

is contained within a single commit.

The point to remember is that each commit should make an easily understood

change that can be verified by reviewers. Each commit should be justifiable

on its own merits.

When dividing your change into a series of commits, take special care to

ensure that CRIU builds and runs properly after each commit in the

series. Developers using `git bisect` to track down a problem can end up

splitting your patch series at any point; they will not thank you if you

introduce bugs in the middle.

## Sign your work

To improve tracking of who did what, we ask you to sign off the commits in

your fork of CRIU or the patches that are to be emailed.

The sign-off is a simple line at the end of the explanation for the

patch, which certifies that you wrote it or otherwise have the right to

pass it on as an open-source patch. The rules are pretty simple: if you

can certify the below:

### Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I

have the right to submit it under the open source license

indicated in the file; or

(b) The contribution is based upon previous work that, to the best

of my knowledge, is covered under an appropriate open source

license and I have the right under that license to submit that

work with modifications, whether created in whole or in part

by me, under the same open source license (unless I am

permitted to submit under a different license), as indicated

in the file; or

(c) The contribution was provided directly to me by some other

person who certified (a), (b) or (c) and I have not modified

it.

(d) I understand and agree that this project and the contribution

are public and that a record of the contribution (including all

personal information I submit with it, including my sign-off) is

maintained indefinitely and may be redistributed consistent with

this project or the open source license(s) involved.

then you just add a line saying

```

Signed-off-by: Random J Developer <random at developer.example.org>

```

using your real name (please, no pseudonyms or anonymous contributions if

it possible).

Hint: you can use `git commit -s` to add Signed-off-by line to your

commit message. To append such line to a commit you already made, use

`git commit --amend -s`.

```

From: Random J Developer <random at developer.example.org>

Subject: [PATCH] Short patch description

Long patch description (could be skipped if patch

is trivial enough)

Signed-off-by: Random J Developer <random at developer.example.org>

---

Patch body here

```

## Submit your work upstream

We accept GitHub pull requests and this is the preferred way to contribute to CRIU.

For that you should push your work to your fork of CRIU at [GitHub](https://github.com) and create a [pull request](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests)

### Pull request guidelines

Pull request comment should contain description of the problem your changes

solve and a brief outline of the changes included in the pull request.

Please avoid pushing fixup commits to an existent pull request. Each commit

should be self contained and there should not be fixup commits in a patch

series. Pull requests that contain one commit which breaks something

and another commit which fixes it, will be rejected.

Please merge the fixup commits into the commits that has introduced the

problem before creating a pull request.

It may happen that the reviewers were not completely happy with your

changes and requested changes to your patches. After you updated your

changes please close the old pull request and create a new one that

contains the following:

\* Description of the problem your changes solve and a brief outline of the

changes

\* Link to the previous version of the pull request

\* Brief description of the changes between old and new versions of the pull

request. If there were more than one previous pull request, all the

revisions should be listed. For example:

```

v3: rebase on the current criu-dev

v2: add commit to foo() and update bar() coding style

```

If there are only minor updates to the commits in a pull request, it is

possible to force-push them into an existing pull request. This only applies

to small changes and should be used with care. If you update an existing

pull request, remember to add the description of the changes from the

previous version.

### Mailing list submission

Historically, CRIU worked with mailing lists and patches so if you still prefer this way continue reading till the end of this section.

### Make a patch

To create a patch, run

```

git format-patch --signoff origin/criu-dev

```

You might need to read GIT documentation on how to prepare patches

for mail submission. Take a look at http://book.git-scm.com/ and/or

http://git-scm.com/documentation for details. It should not be hard

at all.

We recommend to post patches using `git send-email`

```

git send-email --cover-letter --no-chain-reply-to --annotate \

--confirm=always --to=criu@openvz.org criu-dev

```

Note that the `git send-email` subcommand may not be in

the main git package and using it may require installation of a

separate package, for example the "git-email" package in Fedora and

Debian.

If this is your first time using git send-email, you might need to

configure it to point it to your SMTP server with something like:

```

git config --global sendemail.smtpServer stmp.example.net

```

If you get tired of typing `--to=criu@openvz.org` all the time,

you can configure that to be automatically handled as well:

```

git config sendemail.to criu@openvz.org

```

If a developer is sending another version of the patch (e.g. to address

review comments), they are advised to note differences to previous versions

after the `---` line in the patch so that it helps reviewers but

doesn't become part of git history. Moreover, such patch needs to be prefixed

correctly with `--subject-prefix=PATCHv2` appended to

`git send-email` (substitute `v2` with the correct

version if needed though).

### Mail patches

The patches should be sent to CRIU development mailing list, `criu AT openvz.org`. Note that you need to be subscribed first in order to post. The list web interface is available at https://openvz.org/mailman/listinfo/criu; you can also use standard mailman aliases to work with it.

Please make sure the email client you're using doesn't screw your patch (line wrapping and so on).

> \*\*Note:\*\* When sending a patch set that consists of more than one patch, please, push your changes in your local repo and provide the URL of the branch in the cover-letter

### Wait for response

Be patient. Most CRIU developers are pretty busy people so if

there is no immediate response on your patch don't be surprised,

sometimes a patch may fly around a week before it gets reviewed.

## Continuous integration

Wiki article: [Continuous integration](https://criu.org/Continuous\_integration)

CRIU tests are run for each series sent to the mailing list. If you get a message from our patchwork that patches failed to pass the tests, you have to investigate what is wrong.

We also recommend you to [enable Travis CI for your repo](https://criu.org/Continuous\_integration#Enable\_Travis\_CI\_for\_your\_repo) to check patches in your git branch, before sending them to the mailing list.