Make a PR to batocera.linux

We recommend anyone who wishes to contribute to Batocera to join the Discord server for any questions not covered here.

Recommended reading:

- Git has an overview of the broad concepts on their about page.
- Microsoft (the parent company of Github) has a pretty good tutorial for making a pull request (obviously, change Microsoft's documentation Github to Batocera's); this covers most of the same ground that this article does.

Basics

Batocera is an open-source software, its source code is hosted at https://github.com/batocera-linux/batocera.linux. In order to contribute to the project, you must make a pull request to this online repository (a “remote”). You can achieve this by installing a source control manager (the most obvious one being git), make the changes on your local clone and then make a pull request to Batocera's remote (from here on, we will refer to Batocera's remote as “upstream”).

This article mainly pertains to Linux distributions, but this should also be possible with Git Bash on Windows (with some workarounds, read below).

Github also has a desktop GUI for this, which may be more friendlier to those new to Github and source code editing in general.

Making your first pull request

There are multiple steps to make a pull request:

1. Fork Batocera to your own repository
2. Install git on your machine
3. Clone your fork locally
4. Create a new local branch on your fork
5. Make the modifications locally
6. Build Batocera and test the modifications, fixing any issues that arise
7. Push your commits to the new branch
8. Make a pull request to batocera.linux containing those commits

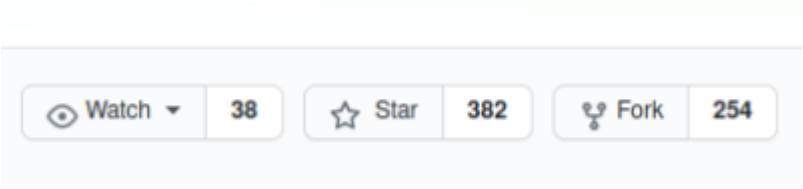
Initial setup involves a few extra steps, but once complete, creating future pull requests are much

easier. All you have to do is make sure your local repository is up to date with upstream.

Fork Batocera to your own repository

To make a Pull Request, you first need to make a fork of batocera.linux:

1. Login to your Github account
2. Go to https://github.com/batocera-linux/batocera.linux
3. Click on the **Fork** button on the right



This will create a fork at `https://github.com/<your Github username>/batocera.linux` for your account respectively. From here on, we will refer to this as the “origin” remote.

Install git

Most Linux-based distributions

The most current instructions are available at Git's official documentation.

To surmise:

- On Ubuntu/other Debian-based distros: `apt-get install git`
- On Fedora: `dnf install git`
- On Arch: `pacman -S git`

If you're on another distro you probably already know how to install `git`.

`git` will typically be available in the official package repository for your distribution, though depending on some distributions this may be an incredibly old version. You can install a later version by adding a PPA:

- On Ubuntu/other Debian-based distros: `add-apt-repository ppa:git-core/ppa` and once successfully executed, `apt update; apt install git`

Other distros generally use more recent versions so this isn't as important.

Windows

Using `git` on Windows is generally not recommended, as it is by default not compatible with objects in the file system such as symbolic links, folders with hidden attributes, line terminators in text files, etc. But if you know what you're doing and are willing to workaround these shortcomings, you can utilize Git Bash.

You'll note that this program also offers “Git GUI”. If you use this, just like with Github Desktop, you're on your own.

Download and install it. There will be multiple questions during installation, with short explanations of what they mean. Choose what you are most comfortable with. For instance, if you are already using Notepad++ to edit text documents, which saves line terminations correctly (compared to Notepad), then you don't need to worry about CR LF to LF conversion (just select “No conversion”). Once installation is complete, you will have Git Bash added to your Start menu. This will launch MinGW, which you can then use `git` commands as you normally would in any regular Bash terminal emulator:

One thing that cannot be worked around is symbolic links, Windows doesn't like them. `git` will, by default, convert symlinks into ordinary files for you. It is recommended to simply not make any changes to any symbolic links in your PRs (you can check which files are symbolic links on the repository).

Not recommended, but you can enable experimental symlink support within Windows by installing the Link Shell Extension and turning off the `symlink = false` in your `.gitconfig`. It is strongly recommended to read through that entire page to be aware of its shortcomings and Windows' general disdain towards symlinks.

The rest of this tutorial can be followed as is if using Git Bash.

Cloning your fork locally

It is important to distinguish between remote and local repositories. Right now, your account has a remote repository that is a fork (copy) of Batocera.linux. We need to clone (download) this origin remote repository to your local computer to be able to make local edits to it.

The easiest way is to open a command line, navigate to the directory you would like to save the origin repository to and run `git clone https://github.com/<your Github username>/batocera.linux`.

Once done (this may take a while, depending on your internet speed), run the following:

- `cd batocera.linux`
- `git remote add upstream`

`https://github.com/batocera-linux/batocera.linux.git`
- `git remote set-url upstream https://github.com/batocera-linux/batocera.linux.git`
- `git remote set-url origin https://github.com/YOURNAME/batocera.linux`
where YOURNAME is your Github username.

This will config the local repository to be recognized as a fork of upstream (the original remote repository). This makes it easier to stay “in sync” with future changes made to upstream.

Create a local branch on your fork

The `master` branch is the “main” branch on which Batocera is built from. If you start making changes directly on your local master branch and make a pull request using that, any additional changes made after the fact **will be added to that very pull request**, preventing you from making additional changes until that pull request has been merged by upstream. Instead, it is advisable to create a new local branch based on the master branch for each set of changes you want to make per pull request, as that will allow you to make more than one pull request at a time.

To create a new local branch, run `git checkout -b name-of-the-branch`. Then you can run `git commit` when you have completed your changes.

`git` has some limitations on what characters you can use for your branch name! Stick to standard alphanumerical characters, and no spaces (hyphens and underscores are accepted).

If you're finding differences where you haven't actually made them, make sure your submodules are updated. `git submodule update --init`.

Making and testing the modifications

You have some guidelines on this page to compile individual packages, a list of notable files and their location on a live install and this page for more general compilation of the whole Batocera Linux system.

Push your commits to the new branch

Once your changes have been tested and you're ready to make a pull request, merge your commit (changes to files) using `git push --set-upstream origin name-of-the-branch`. This will “push” your changes to the origin remote. You can double check that this has occurred by visiting your appropriate branch on your fork at https://github.com/YOURNAME/batocera.linux

Once you've confirmed that you have successfully pushed your changes to your own origin remote,

go to the produced link in the output of the command and click on `Pull Request`. This will let you ask to merge the changes into the main repository.

Making changes in the future

After setting this up, the process for making new pull requests becomes much simpler. Just make your own master branch up to date with the upstream master branch before creating a new branch. To do this, you need to do:

1. `git checkout master` : Tells git to use the master branch
2. `git fetch upstream` : Tells git to grab every changes from the upstream repository
3. `git rebase upstream/master` : Tells git to make your branch up to date with the master branch of the upstream repository

It is not recommended to use `git pull upstream master`. Here's a good blog post about it.

Then repeat the steps from the "Create a new local branch on your fork" section above.