Make a PR to batocera.linux

This page is, currently, far from being perfect. We recommend anyone who wishes to contribute to Batocera to join the Discord server for any questions not covered here.

Basics

Batocera is an open-source software, it's source code can be found at https://github.com/batocera-linux/batocera.linux. In order to contribute to the code, you can make a pull request with the changes you made to the OS.

Microsoft (the parent company of Github) has a pretty good tutorial for this using Git Bash on Windows: https://docs.microsoft.com/en-us/contribute/get-started-setup-local

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. Make the modifications locally
5. Build Batocera and test the modifications, fixing any issues that arise
6. Create a new local branch on your fork
7. Push your commits to the new branch
8. Make a pull request to batocera.linux containing those commits

The first time, you will need to fork the system and clone it locally. The next time, you will simply need to make your local copy up to date with the upstream branch.

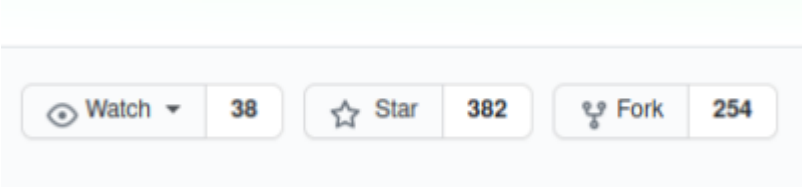
Github also has a desktop GUI for this, however instructions for this client cannot be provided (yet). If you choose to go down this route, hopefully the GUI is intuitive enough to work out what the equivalent actions on it are.

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` at your account respectively.

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. It is strongly recommended to read through that entire page to be aware of its shortcomings and Windows' general disdain towards symlinks.

Cloning your fork locally

You have to clone this fork locally, the easiest way is to open a command line and type `git clone https://github.com/<your Github username>/batocera.linux`.

You then need to go into the created folder (which should be `batocera.linux`), once inside, you will need to tell it this is a fork of https://github.com/batocera-linux/batocera.linux using the following commands:

- `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`

Finally, you have to tell it the URL of your own fork itself using :

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

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.

Create a local branch on your fork

You then need to make a local branch using `git checkout -b Name-of-the-branch` , once this is done, you have to commit your changes to it using `git commit`.

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

Push your commits to the new branch

Once all of this is done, commit them to your branch on your fork using `git push --set-upstream origin Name-of-the-branch`

Making the pull request

You can now go into your Github fork link, and click on `Pull Request`. This will let you ask to merge the changes into the main repository.

Make new changes

Once you have set this up, you can simply make your own master branch up to date with the upstream master 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.

You then make your changes, create a new branch using `git checkout -b Name-of-the-branch` and commit your changes to it like before. The rest doesn't change from the first time.