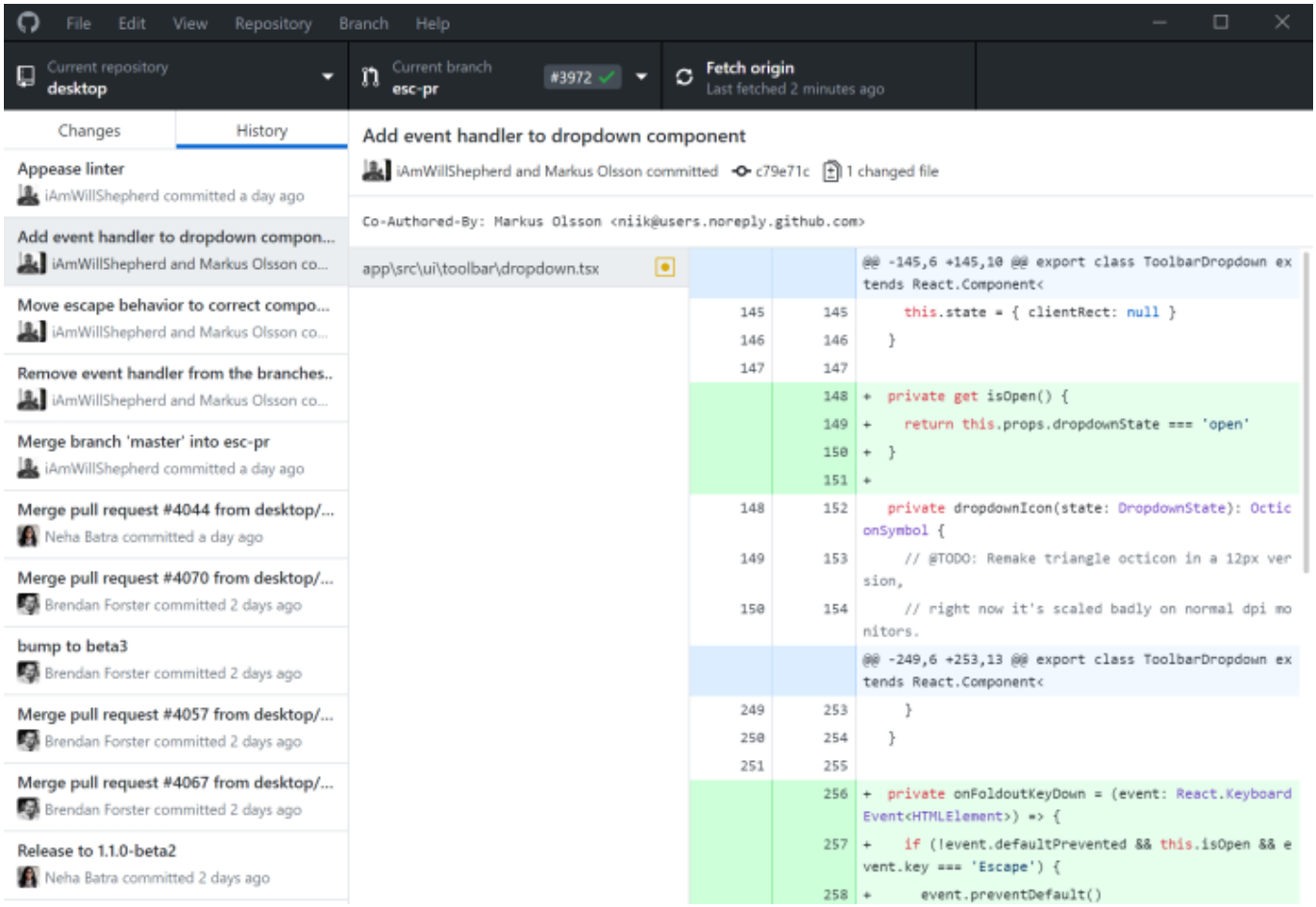
Github Desktop

Github released a neat GUI for managing local repositories with `git` called Github Desktop. It's experimental but in a pretty functional state if you choose to use it, it can be a great learning tool if you're new to contributing to open-source code. With that being said, if you want to do more advanced or technical stuff, you'll probably want to switch over to using the `git` command line tool in the future as it offers much more functionality.



Github Desktop is only confirmed to be working for Github-hosted repositories, which is what Batocera uses. This may not work as equally for other repositories hosted on other services.

How to install

Windows

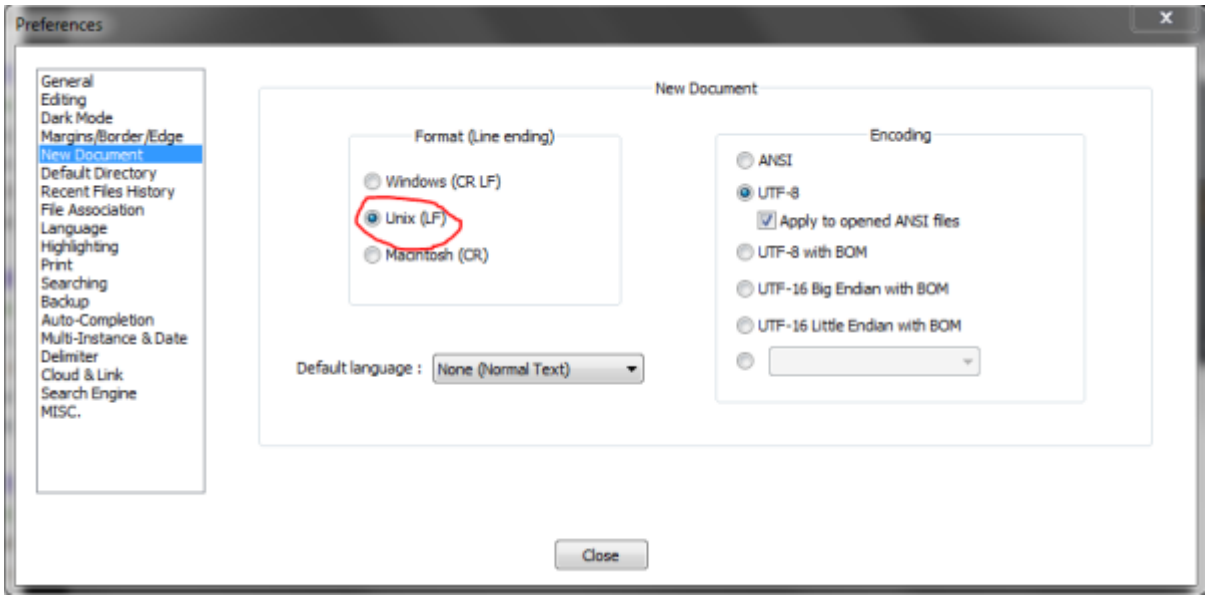
Easy, just go to Github Desktop's download page and follow their instructions.



Note for Windows users

Windows users should take some extra precautions when editing Linux-spawned files. NTFS, the default filesystem for Windows, does not support many features that the more modern filesystems Linux supports such as symlinks, file attributes (including the extremely important executable bit), etc. By default, Git will do its best to transparently convert file attributes to and from Windows-acceptable formats to the ones on the remote repository, but edits are not always possible (such as with two symlinked files).

Windows' built-in text editing software doesn't support the line terminators used in Linux text files. Install a good text-editing software such as Notepad++ to work around this; make sure to set the correct line terminators in **Settings → Preferences… → New Document → Format (Line ending)** to “Unix (LF)”:



Then also tell `git` to not automatically convert LF to CRLF by adding the following to your `C:\Users\<your-name>\.gitconfig` file:

```
[core]
    autocrlf = input
```

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) and adding the following to your `C:\Users\<your-name>\.gitconfig` file:

```
[core]
    symlinks = true
```

If you already have a repository setup and are simply transitioning over to Github Desktop, you'll also need to erase the `symlinks = false` from the `<repository-name>/.git/config` file in the repository folder itself.

Once this is done, change Github Desktop's shortcut to always launch in administrator mode, as that will allow it to create and edit symlinks.

Mac

Easy:

1. Go to Github Desktop's download page
2. Click “Download for macOS” and extract the downloaded ZIP archive
3. Double-click `GitHub Desktop` from the extracted files
4. Follow the on-screen instructions

Linux-based distributions

Github Desktop does not official support any Linux-based distributions, but a community fork has been released at Shiftkey's Github Desktop repository. The readme contains the most up-to-date installation instructions for most major distributions.

In the case that you are using a distribution that cannot install `deb` or `rpm` packages, an AppImage is available on the releases page.

On that note, since Github Desktop uses Electron, it doesn't have that many dependencies. It may just be possible to simply copy + paste the package's contents from the `rpm` package into a random folder and just execute it from there.

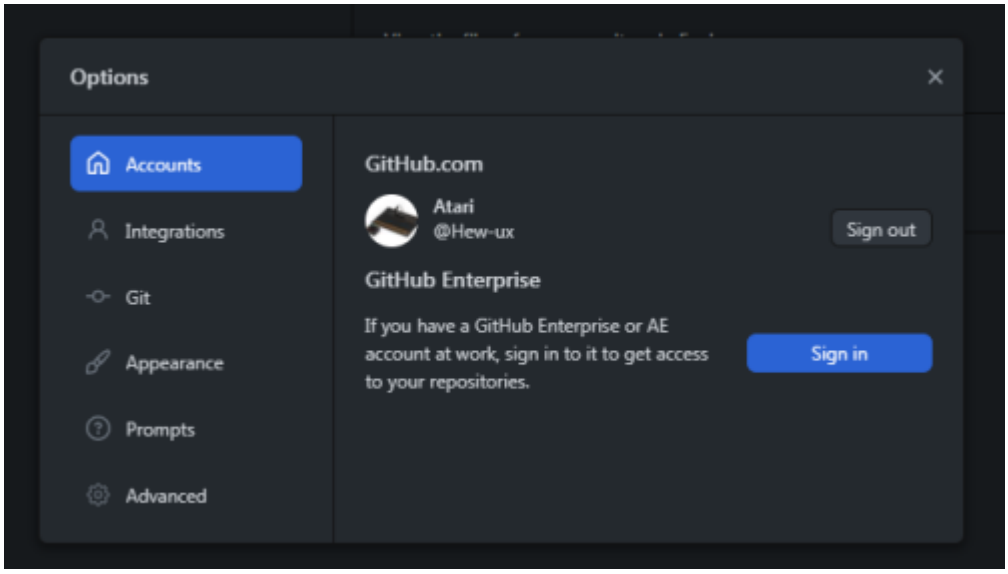
On Solus, using the AppImage is recommended. Download it and mark it as executable. Installing Chrome (from the third party section in the software updater) is required to authenticate the sign-in. You don't actually need to use Chrome itself to do the authentication, it just needs to be installed. Yeah.

Setting up Github Desktop

Sign in

After launching Github Desktop for the first time, you should sign into your Github account. In case the dialogue didn't automatically appear, you can do this by going to **File → Options → Accounts** and clicking **Sign in** in the “Github.com” section.

Once signed in, it should look like this:



If having difficulty getting your browser to recognize the protocol (either due to weird permission issues or manual installation), reboot your computer and try again. Seriously. Github Desktop installs the protocol handlers which only get activated on boot for some operating systems.

If it's still failing after that, it is possible to copy and paste a known-working configuration folder from another operating system into the one you are having the difficulties with.

- Windows config folder: `Users\<Your username>\AppData\Roaming\GitHub Desktop`
- Linux config folder: `~/.config/GitHub Desktop`

Make sure GitHub Desktop is closed before doing this.

Fixing submodule downloading

Batocera makes ludicrous use of submodules to compile. You'll find that when attempting to make new branches, they'll default to an older version of the submodule than the one that's currently being used. To work around this, add the following lines to your global gitconfig (on Windows, this is at `C:\Users\<your name>\.gitconfig`; on Linux, this is at `~/.gitconfig`):

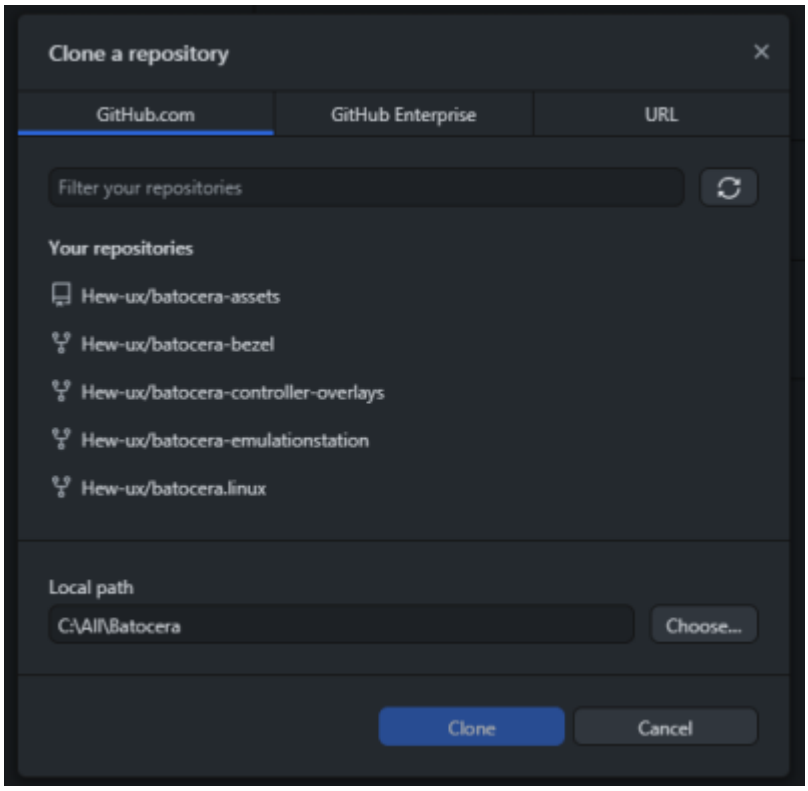
```
[submodule]
    recurse = true
```

This can also be achieved by running `git config --global submodule.recurse true` from the currently installed instance of `git`.

In a typical Windows installation of Github Desktop, this is at `C:\Users\<your name>\AppData\Local\GitHubDesktop\app-<#.#.#>\resources\app\git\cmd\git.exe`. Navigate to that folder, then press `[Shift]` + right-click and press **Open command window here**, then run `git config --global submodule.recurse true`.

Clone the repository

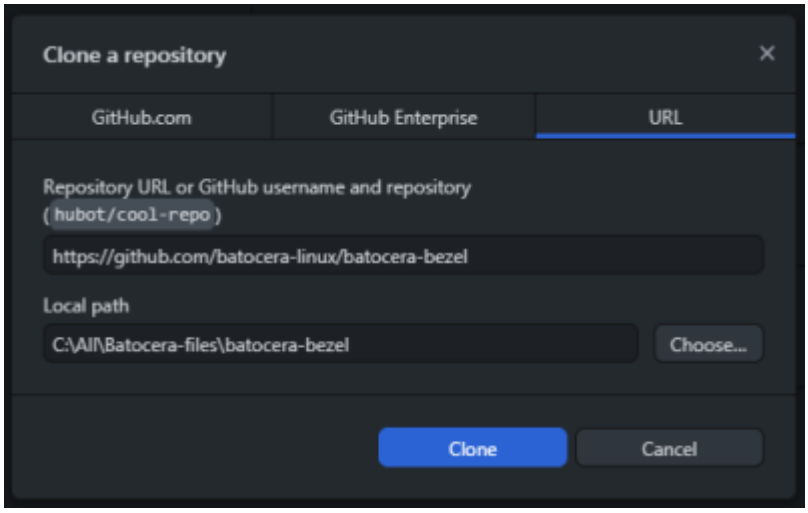
Go to **File** → **Clone repository...**. This will open up a dialogue window:



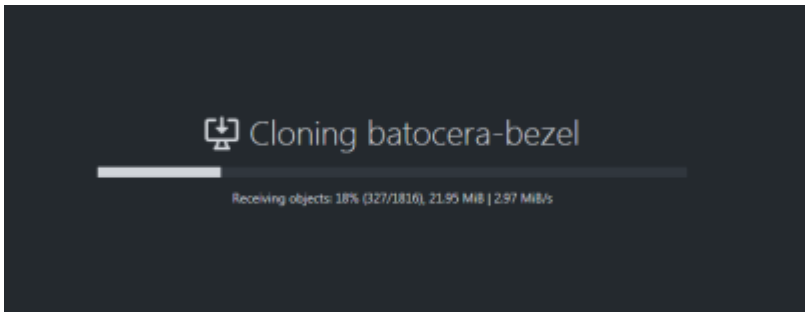
If you've already forked the project, great! Just click it here and clone it, choosing to contribute to the parent project.

Otherwise, click on **URL** at the top to switch to another screen.

Enter the URL of the repository you want to clone. Here, I will be copying the batocera-bezels repository, but the process is identical for batocera.linux as well. This is what it should look like:



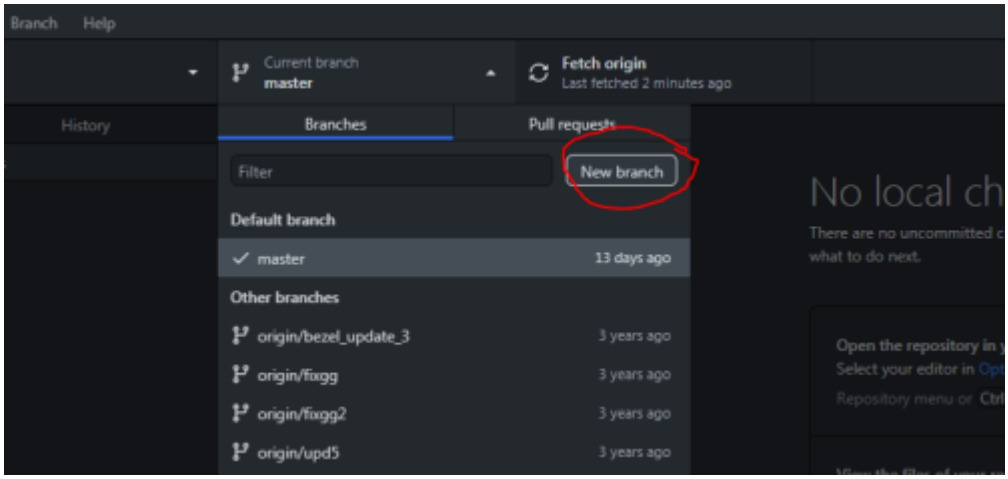
Select your desired folder and click **Clone**. Please wait patiently while the repository is downloaded to your computer.



When asked what project you'd like to contribute to, select the one hosted by batocera.linux, not recalbox or your own account.

Create a new branch

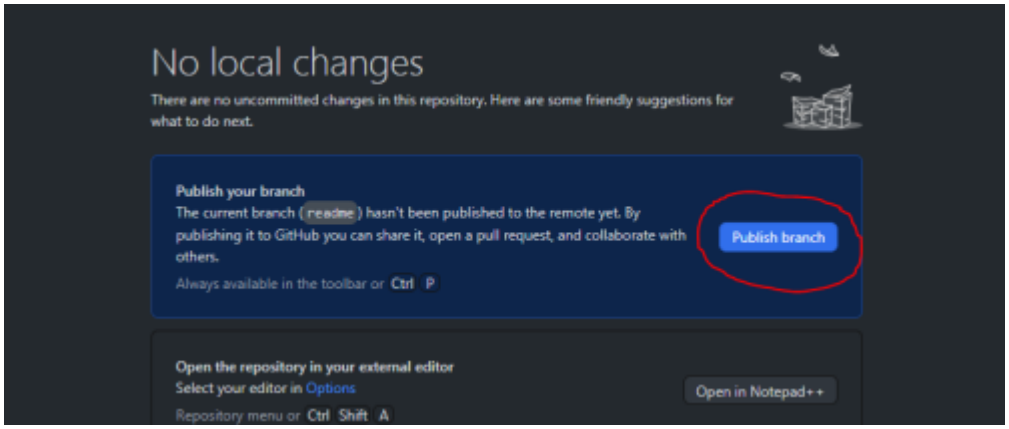
Nice! Now we have the repository on our computer. The default branch is `master`. In order to make our changes, we must first create a new branch. Click on the **Current branch** at the top, then click **New branch**:



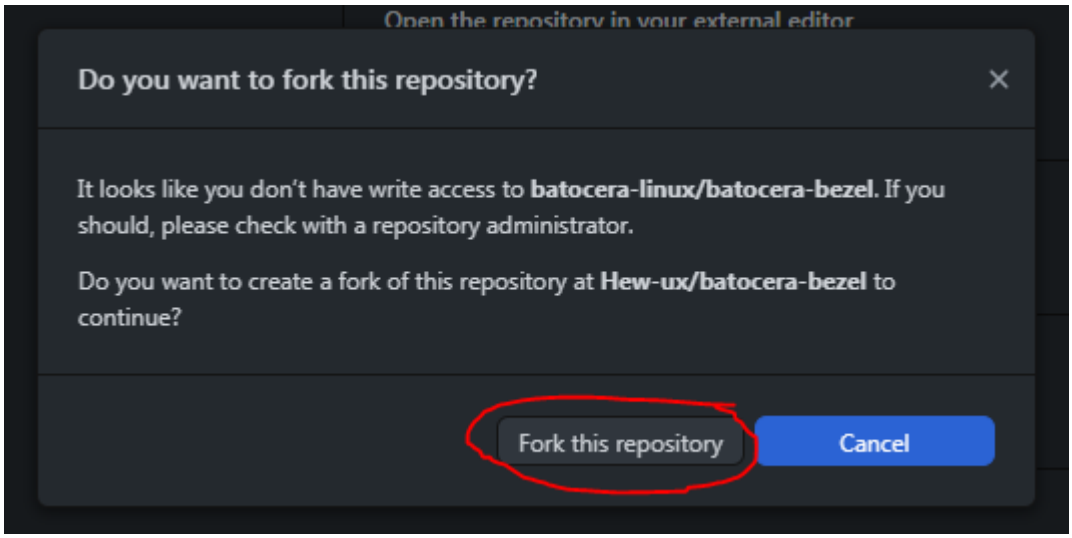
It's useful to give the branch a meaningful, one-or-two word name. I'm intending on making edits to the readme file, so I'm going to name my branch `readme`.

Note that branch names are limited to alphanumeric, hyphen and underscore characters, and no two branches can have the same name; Github Desktop will inform you of this and make the appropriate changes as needed.

Now this branch is stored locally on the computer, but it doesn't exist on the online remote repository. To fix that, click **Publish branch** in the main section:

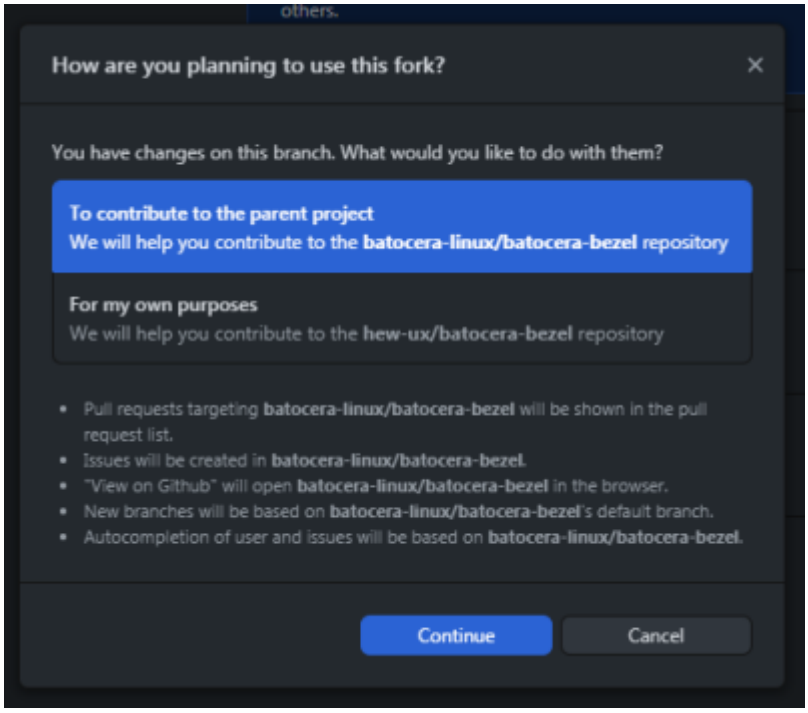


But you'll notice that this will partially fail as you don't have write access to the Batocera-owned repository! You'll be offerred to create your own fork of the repository to make your edits to, click **Fork this repository**:

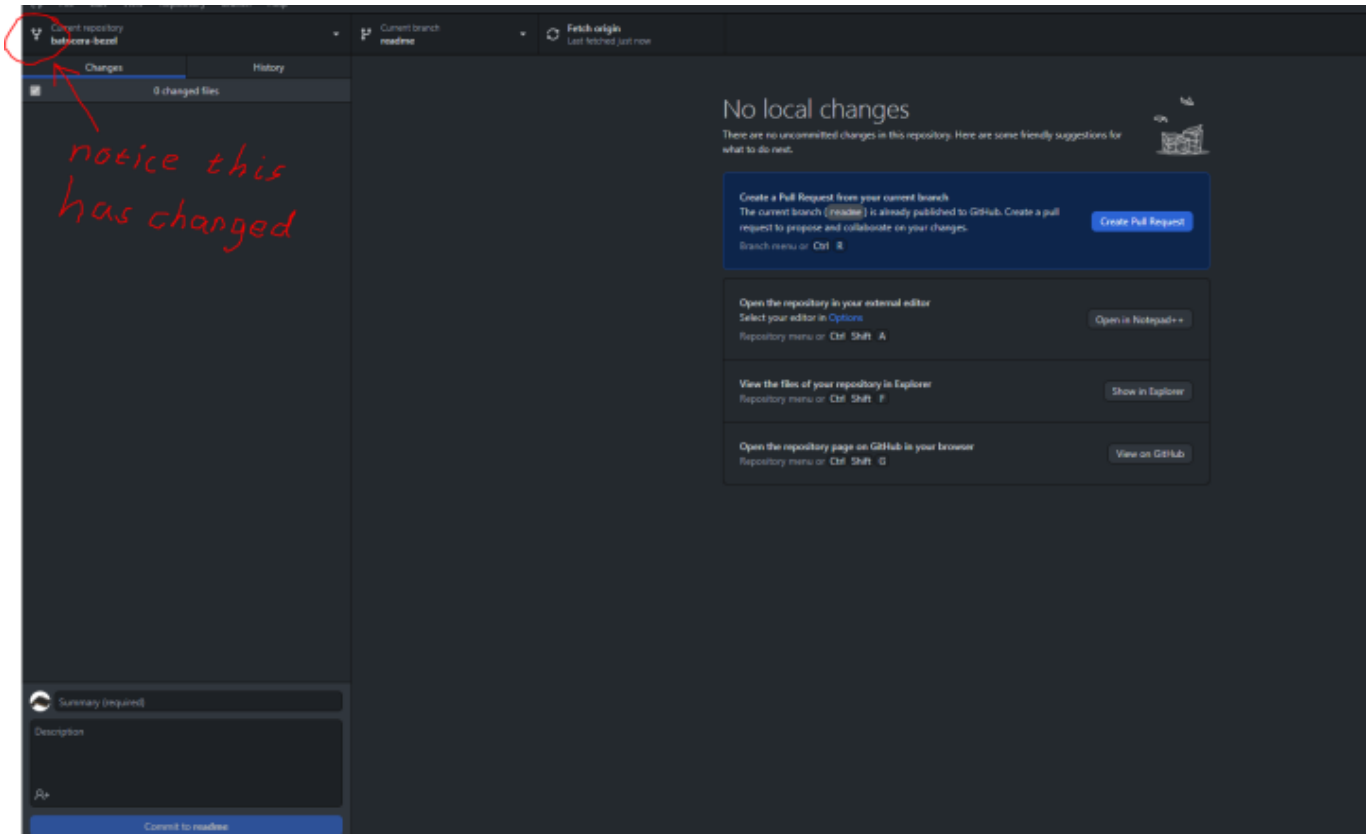


If you're cloning an already forked project, this step will not occur.

You'll then be asked how you'd like to use the project. Leave it on **To contribute to the parent project** pointed at batocera.linux and click **Continue**:



You'll be taken back to the main page. Click **Publish branch** again, and this time everything will be successful!



Your files are now ready to be edited.

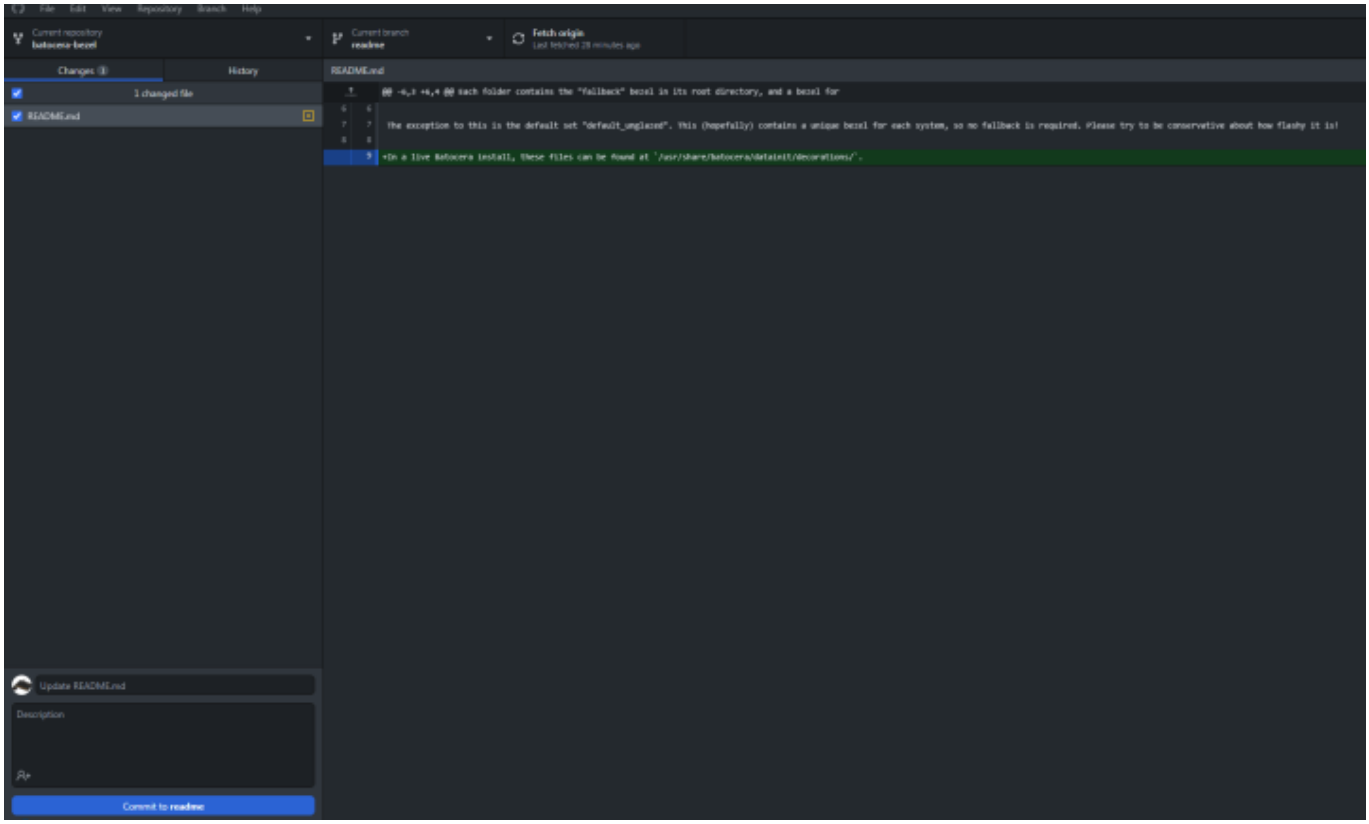
Editing files

Click on **Show in Explorer** (or the name of your preferred file manager if you have something different from default) to open up a folder directly to your repository and start editing away! You have

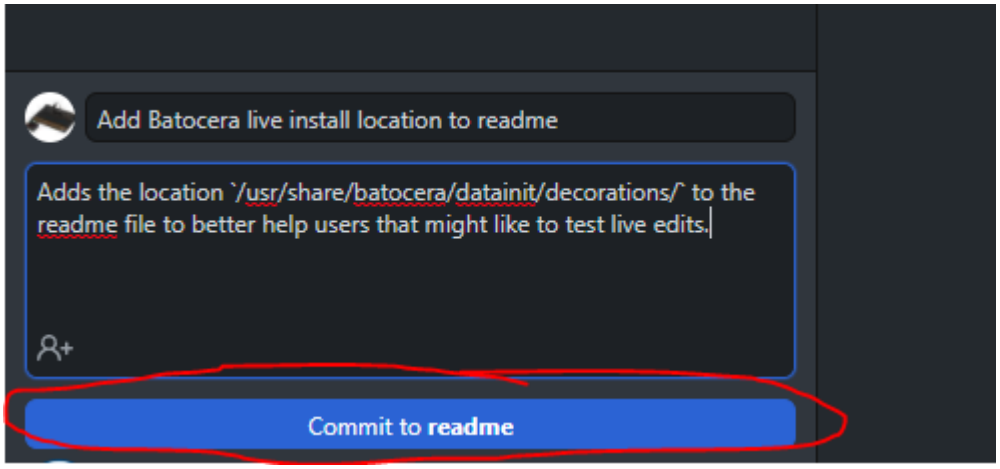
some guidelines on this page to compile individual packages, a list of notable files and their location on a live install and this page for more general compilation of the whole Batocera Linux system.

Committing changes

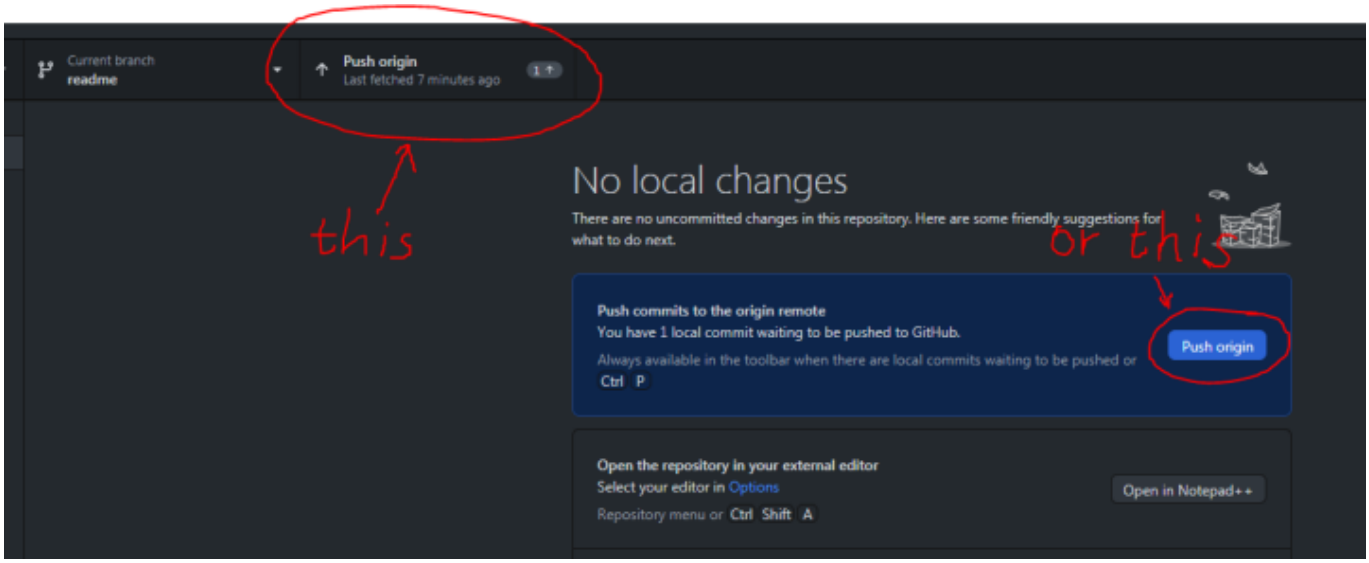
You'll notice that as you edit files, the files you edit and changes you've made will appear in Github Desktop:



You can make as many edits as you'd like. Once you are ready to commit your changes to the fork, fill out the details of the changes you've made in the description box at the bottom left of the screen and click **Commit to <branch-name>**.

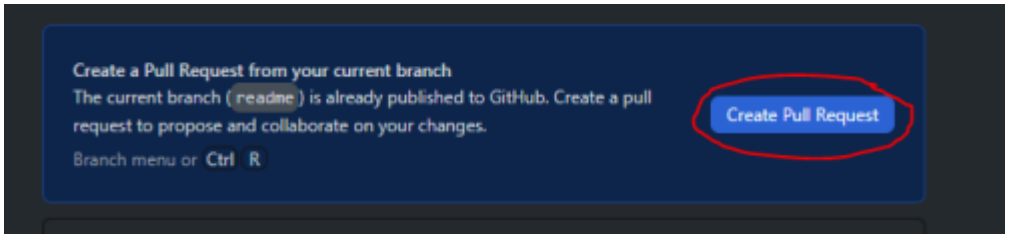


The commit is now saved locally to your repository, but has yet to be pushed to your remote fork. Click on **Push origin** to do so:



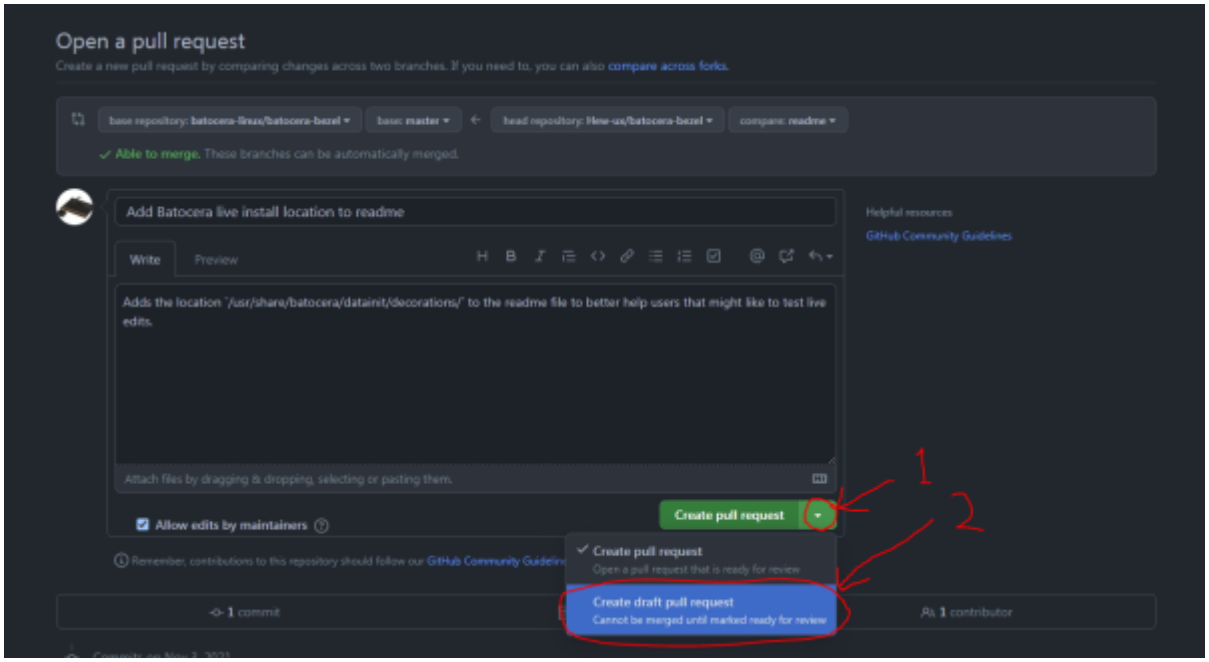
Make a pull request

Once your changes are committed, click on **Create Pull Request** to open your web browser to make a pull request on Github itself.

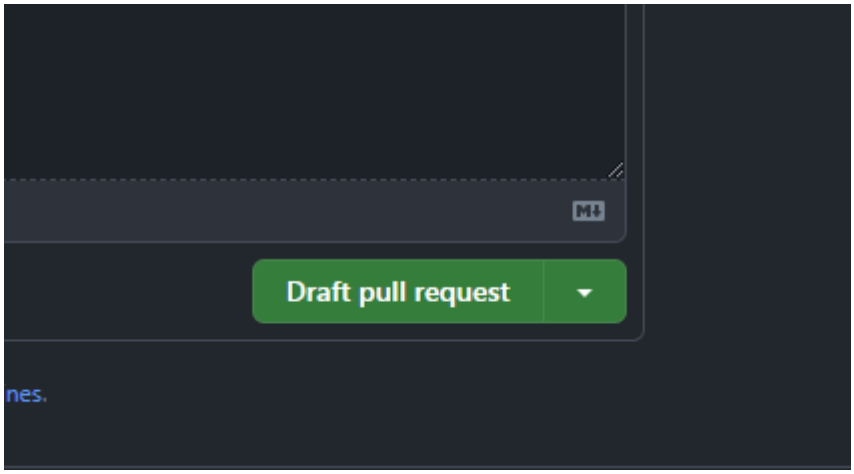


Github is smart enough to automatically insert the title and description you entered for the commit you made earlier, but do note that this is only for the first commit you made. If you've made multiple commits, make sure to double check the fields here.

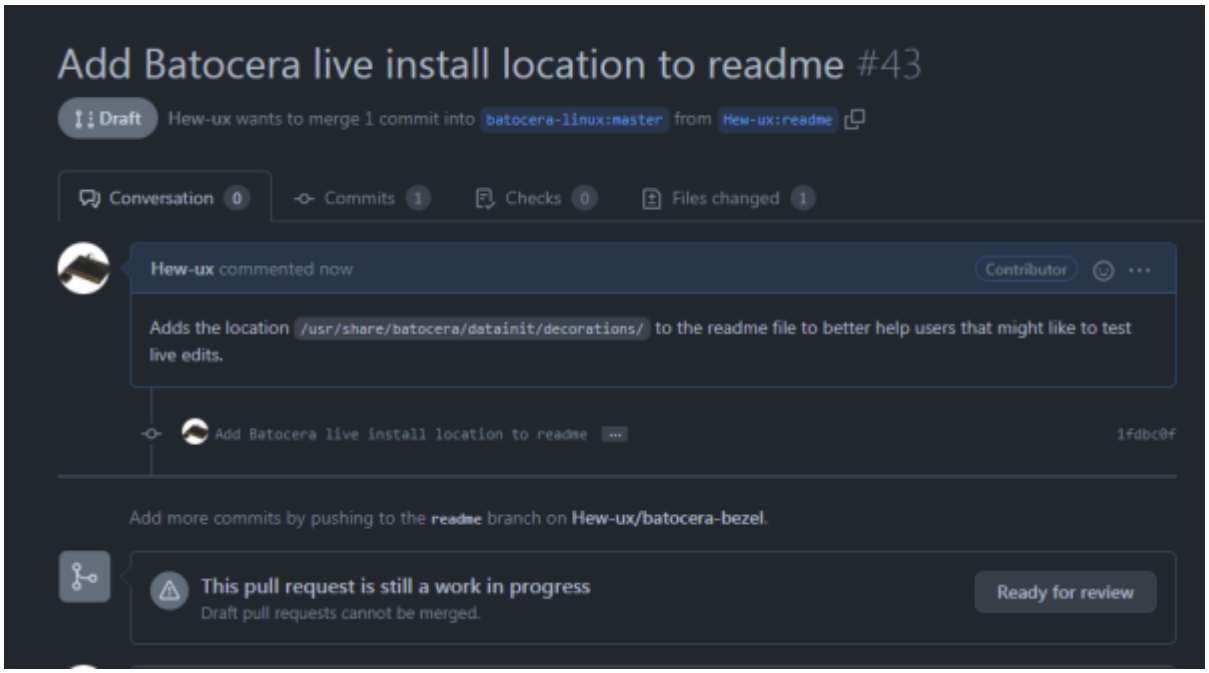
It is recommended to switch this to a **Draft Pull Request** until you've become more familiar with the process. This will allow you to make mistakes without worry until you click **Ready for review**. First click the down arrow and select **Create draft pull request**:



You'll notice that the button now says “Draft pull request”. Click it to continue:



Yay!



A pull request can only be done via the web page, there is no way to do this from Github Desktop itself. This is by design.

When you're ready to submit your changes, click **Ready for review**.

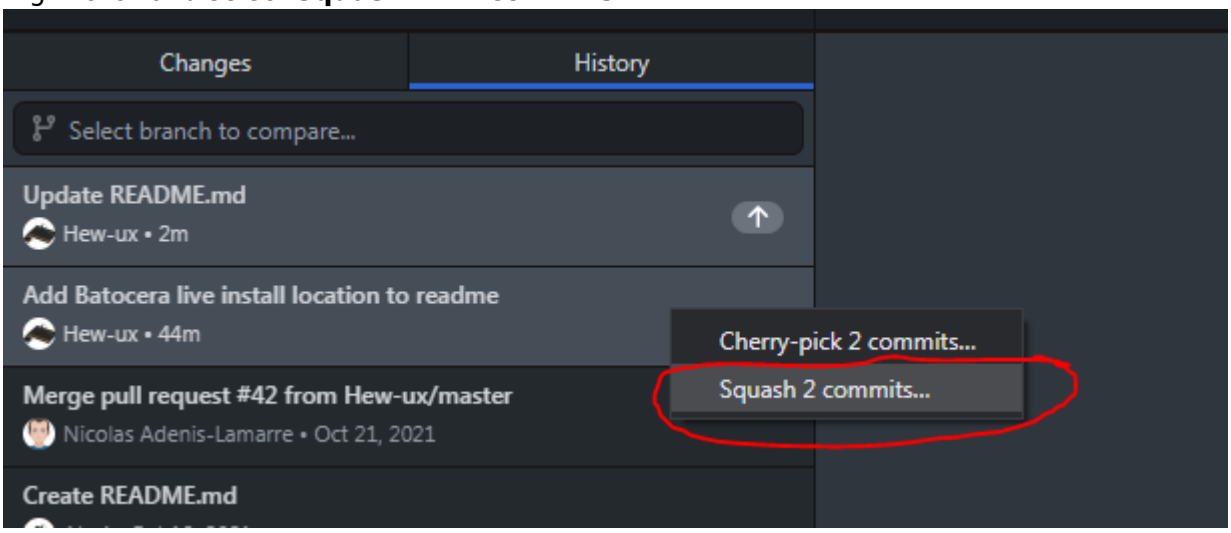
I messed up and need to change something

Don't fret, you can still make changes to your pull request *as long as it hasn't been merged yet* (a draft pull request indicates it is not ready to be merged yet). Ensure that you're on the branch that you made the pull request from, make your additional changes, and commit them again. The changes will be reflected in the pull request immediately.

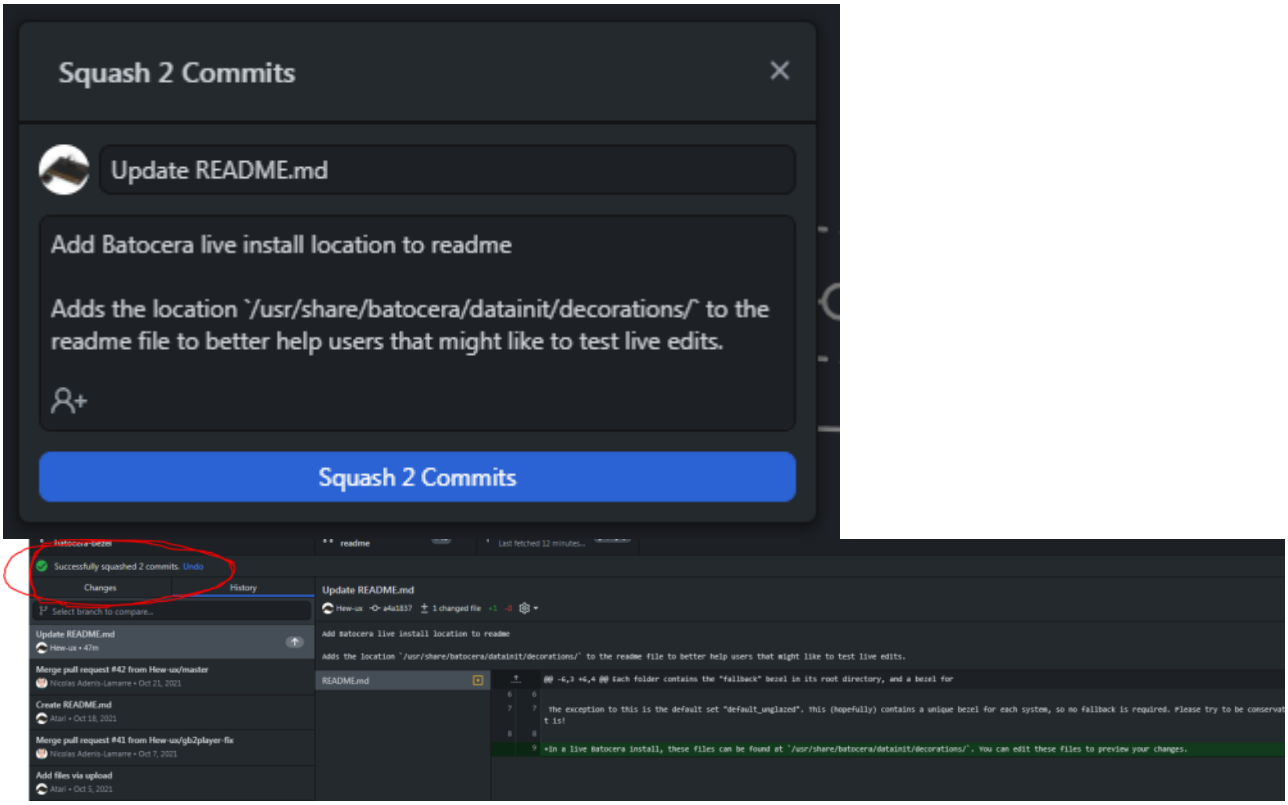
If you've ended up making many commits, it is polite to “squash” them into one commit as it is easier to keep track of things that way. After all, if you're a dev and find that a component of Batocera has broken due to a change made months ago, sifting through ten or twenty commits titled “fixed mistake” in the commit history with little context as to what they were for isn't the most pleasant experience.

To squash multiple commits into one:

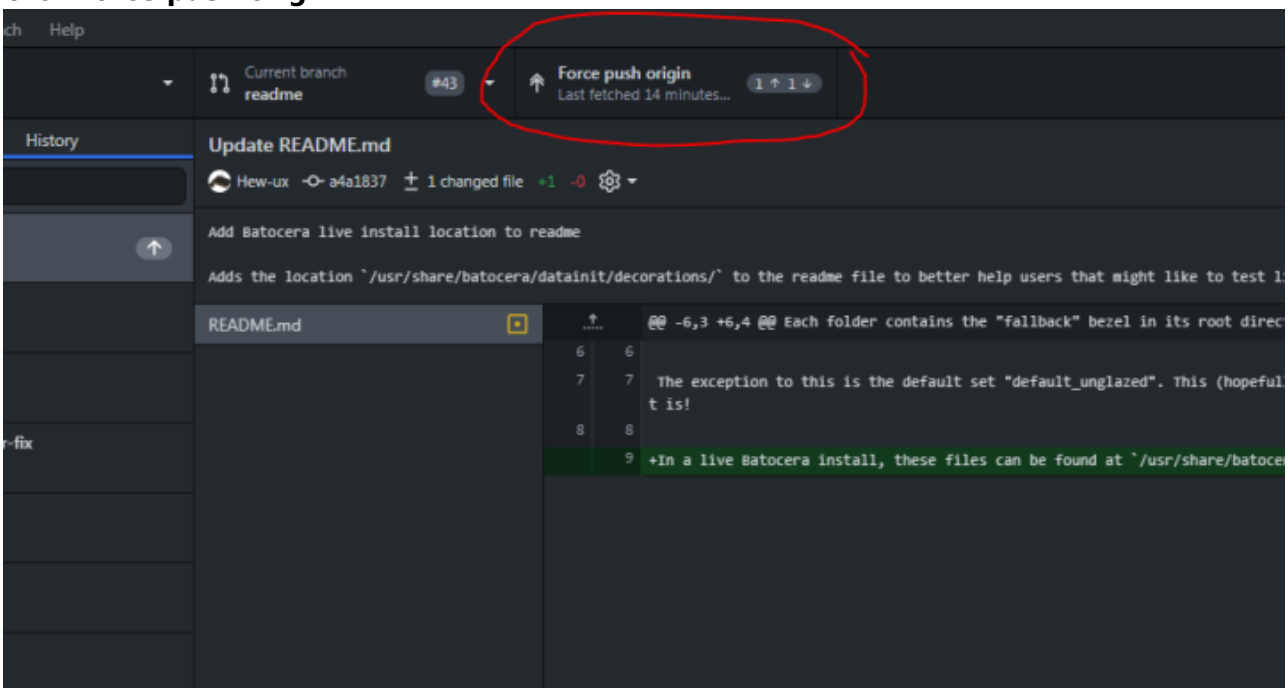
1. Ensure you're still on the branch you have made the pull request on
2. Click on **History** near the top left of the window
3. Select the commits you wish to squash (either by holding `[Ctrl]` and clicking each one or holding `[Shift]` and selecting the first commit you made)
4. Right-click and select **Squash <#> commits**



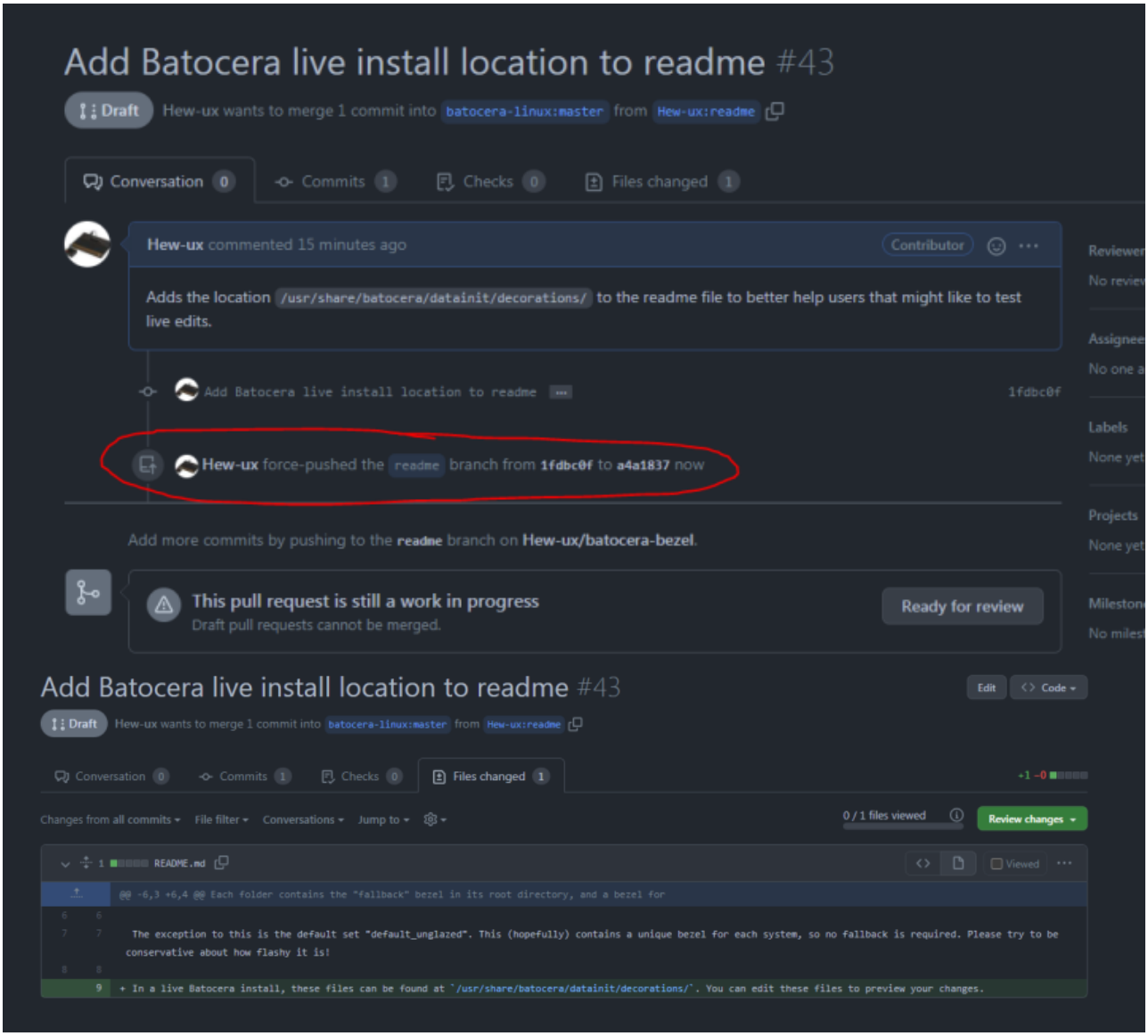
5. Create a new title and description that includes all your changes



6. Click **Force push origin**

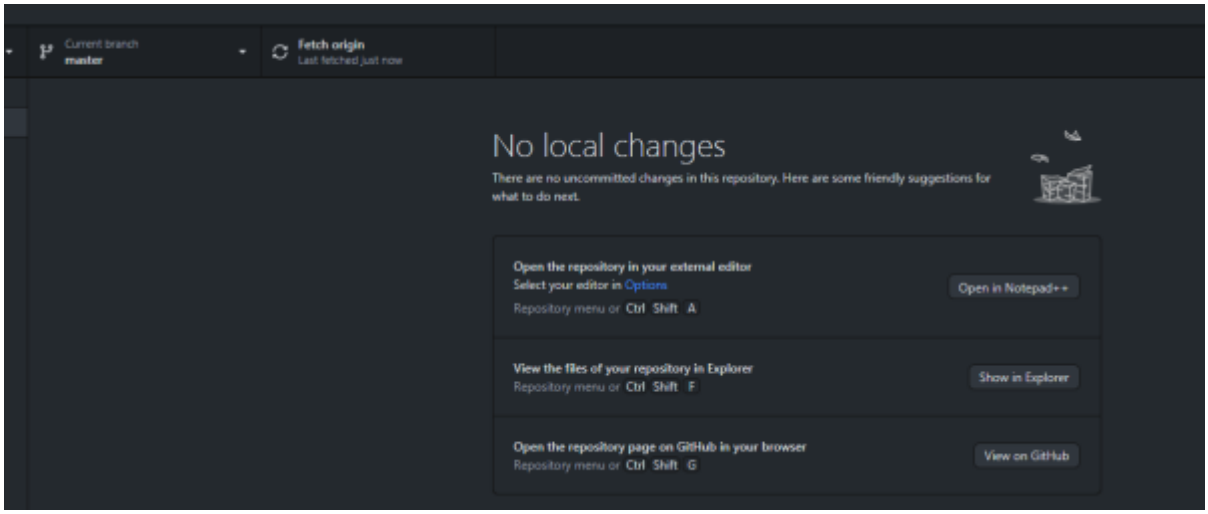


7. Confirm the changes have been reflected in the pull request

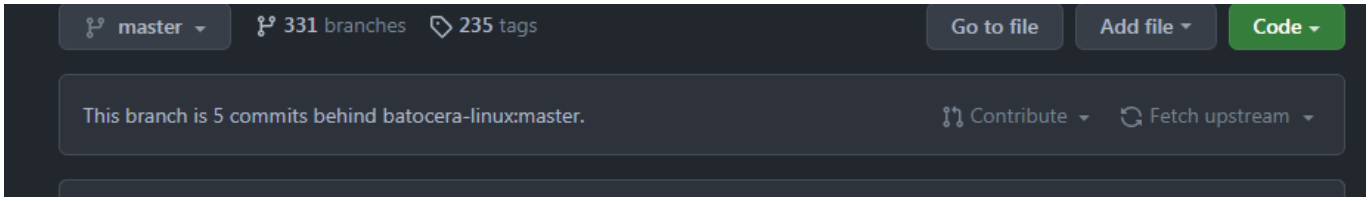


Fetch upstream changes

In order to keep your local master branch in sync with changes made to the upstream master branch, you'll have to fetch the changes. Normally, you would just click **Fetch origin** at the top to do so, but if you do that you'll notice something after a while. On your Github Desktop, it will claim that there are no changes:



But if you visit the fork on your Github webpage:



To repair this, there are multiple methods:

Just use batocera.linux's master branch from upstream

This is the simplest solution, just use batocera.linux's `upstream/master` branch instead. However, if you attempt to do this, you'll be met with a “Fatal: this branch already exists” error. This is because your fork's master branch has the same name. This can be worked around by either:

- Renaming the `master` branch to something unique; or
- Deleting the `master` branch locally from your computer

To do the latter, first be on a branch that isn't master, then right-click the local `master` branch (it shouldn't have anything ahead of it like `upstream/` or `origin/`) and choose to delete it. If you are offered to delete the branch on the remote as well, decline (what's on the remote doesn't truly matter for our purposes here). You'll then be able to switch to `upstream/master` with no issues.

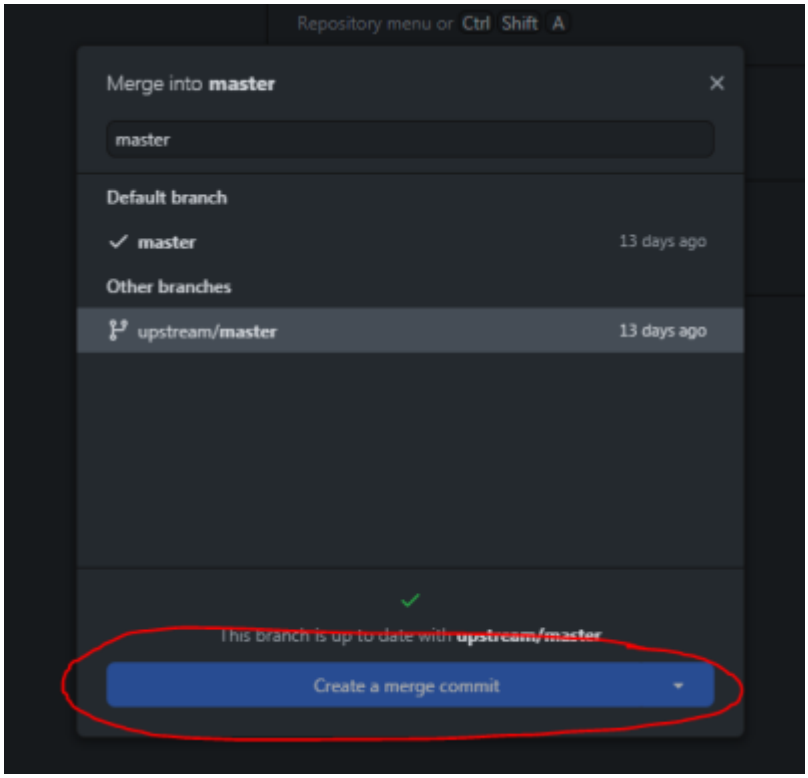
Merging/rebasing the master branch

Similar to the command line procedure, we will merge directly from upstream. If you choose to use this method, this needs to be done every time you want to sync with upstream.

Using Github Desktop

Merge the upstream/master branch to your own master branch clicking **Current branch** → select the “master” branch of your fork → **Current branch** (again) → **Choose a branch to merge into master**. That will bring the merge window up. Choose “upstream/master” from the list and **Create a**

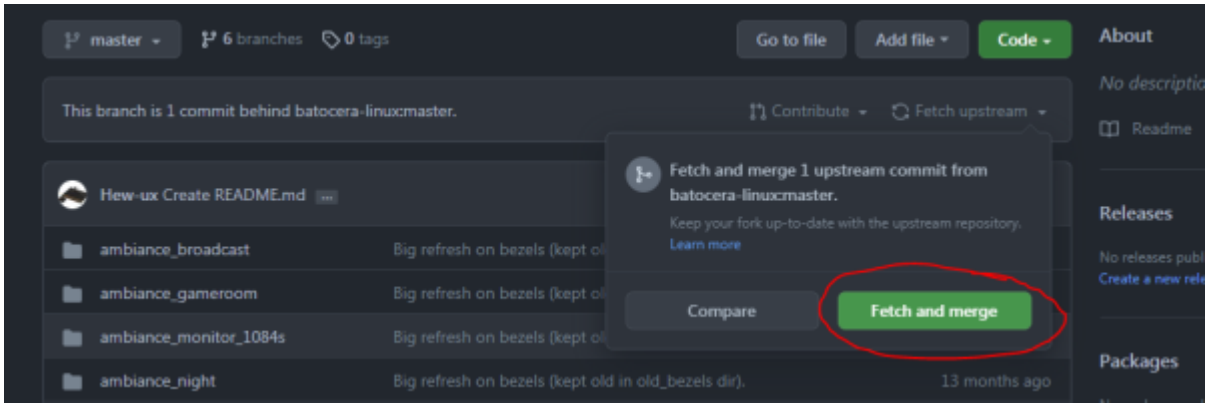
merge commit (or if you want a “cleaner” commit history, **Rebase**):



Then push your changes to your remote fork by pushing the commits. This will be your “fetch” method.

Using Github webpages

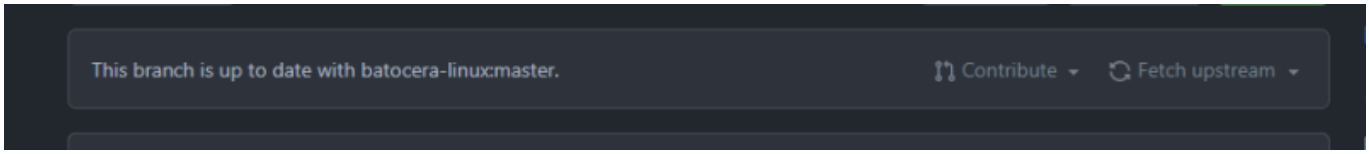
Navigate to `https://github.com/<your name>/batocera.linux`, select **Fetch upstream** → **Fetch and merge**:



You will need to fetch the changes from Github Desktop after doing this.

Confirm fork has been updated

You can check that the remote is up to date by checking your fork online at Github:



Redirecting the local master branch header

Instead, you can edit the branch settings for your `master` branch to instead fetch from upstream instead of your fork (origin). This only needs to be done once.

Change the currently tracked upstream branch by running the command `git branch --set-upstream-to=upstream/master`.

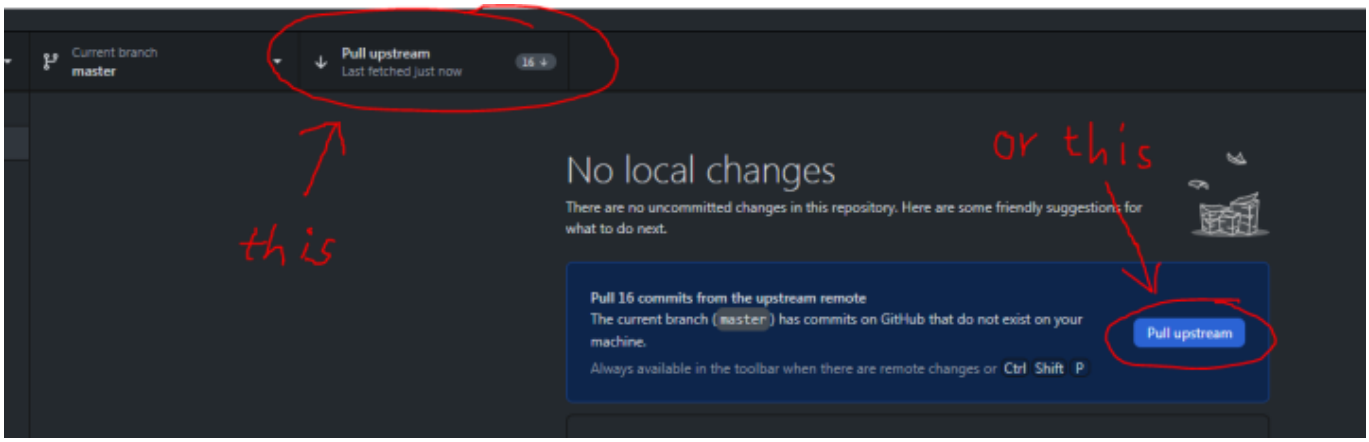
On Windows:

1. Open `C:\Users\<your name>\AppData\Local\GitHubDesktop\app-<#.#.#>\resources\app\git\cmd\`
2. Press `[Shift]` + right-click and click **Open command window here**
3. Run `git -C "C:\<path\to\repository>" checkout master`
4. Then run `git -C "C:\<path\to\repository>" branch --set-upstream-to=upstream/master`

If you use this method, your fork online will not be updated in sync with your local branch. This does not matter, as all your new branches will be based on upstream/master, so any changes will just not be reflected on your online fork. You can always update using Github webpages to keep the online repository up to date as well.

Fetch the changes

After updating the fork online or editing the branch settings, simply click **Pull upstream**/**Pull origin** at the top of the window:



Making new changes

Now that you've got everything set up making new changes in the future is much simpler:

1. Fetch the latest changes made upstream
2. Make a new branch
3. Make your edits to the files
4. Commit those changes to your fork
5. Make a pull request

Troubleshooting

Github Desktop is still in an experimental state and sometimes issues with usage crop up. Their documentation should answer most basic questions.

I can't sign in/authentication fails/can't push commits to my remote

Try resetting your username and authentication token by going to **File** → **Options** → **Accounts** → **Sign Out**, then signing in again. You may also need to explicitly sign out of Github on your web browser to get it to regenerate a new, valid token.

Other/more complicated issues

Refer to Shiftkey's Known Issues page to see if there's a known workaround. There is also a similar page on Github Desktop's official documentation. Unfortunately Batocera itself cannot offer much support for using Github Desktop; if you're having too many issues with Github Desktop it's suggested to switch over to using the Git command line tool instead.

From:
https://wiki.batocera.org/ - **Batocera.linux - Wiki**

Permanent link:
https://wiki.batocera.org/github-desktop?rev=1660093325

Last update: **2022/08/10 01:02**