Submitting GitHub Pull Requests to FreeBSD

BY WARNER LOSH

The FreeBSD Project recently started supporting GitHub pull requests (PRs) to make it easier to contribute. We found that accepting patches via our bug tracker Bugzilla resulted in far too many useful contributions being ignored and growing stale, so contributors should prefer GitHub PRs for changes, leaving bugs in Bugzilla. While Phabricator works well for developers, we've also found it's easy to lose track of changes from outside contributors there. Unless you are working directly with a FreeBSD developer who has told you to use Phabricator, please use GitHub instead. GitHub PRs are easier to track, easier to process, and more familiar to the wider open source community. We hope for faster decisions, fewer dropped changes, and a better experience for all.

Since FreeBSD's volunteers have limited time, The Project has developed standards, norms, and policies to use their time efficiently. You'll need to understand these to submit a good PR. We have some automation which helps submitters fix the common mistakes, allowing the volunteers to review nearly ready submissions. Please understand we can only accept the most useful contributions and some contributions cannot be accepted.

We hope pull requests make it easier to get changes into FreeBSD and provide quick decisions when there are issues.

Next, I'll cover how to turn your changes into a Git branch, how to refine them to meet the FreeBSD Project's standards and norms, how to make a PR from your branch, and what to expect from the review process. Then I'll cover how volunteers evaluate PRs and tips for perfecting your PR.

This article focuses on commits to the base system, not the documentation or ports trees. These teams are still revising the details for these repositories.

Project Standards

The Project has detailed standards for various aspects of the system. These standards are described in the FreeBSD Developer's Handbook and the FreeBSD Committer's Guide. Coding standards are documented in FreeBSD manual pages. By convention, manual pages are divided into sections. All the style manual pages are in section 9 for historical reasons. References to manual pages are traditionally rendered as the name of the page, followed by its section number in parentheses, for example style(9) or cat(1). This documentation is available on any FreeBSD system with the man command, or online.

The Project strives to produce a well-documented integrated system that covers both a kernel that controls the machine as well as a user-space implementation of common Unix utilities. Contributions should be well written with relevant comments. They should include updates to the relevant manual pages when the behavior changes. When you add a flag to a command, for example, it should be added to the manual page as well. When new functions are added to a library, new man pages should be added for the functions. Finally, The Project views the metadata in the source code control system part of the system, so commit messages should conform to the project's standards.

The Project's Standard for C and C++ code is described in style(9). This style is often referred to as "Kernel Normal Form" and is adopted from the style used in Kernighan and Ritchie's *The C Programming Language*. It's the standard that research unix used, as continued within the CSRG at Berkelely who produced the BSD releases. The FreeBSD project has modernized these practices over the years. This style is the preferred style for contributed code, and describes the style used in most of the system. Contributions which change this code should follow this style except for a few files that have their own style. Lua and Makefiles also have their own standards, found in style.lua(9) and style.Makefile(9) respectively.

Contributions should be well written with relevant comments.

Commit messages follow the form favored by the Open Source communities that use git. The first line of the commit message should summarize the entire commit, but do so in 50 or so characters. The rest of the message should describe what changed and why. If what changes is obvious, only explaining why is preferred. The lines should be 72 characters or fewer. It should be written in the present tense, with an imperative tone. It ends with a series of lines that Git calls "trailers" which The Project uses to track additional data about the commits: where they came from, where details about the bug can be found, etc. The Commit Log Message section of the Committer's Guide covers all the details.

Unacceptable Changes

After a few years of experimenting with accepting changes via GitHub, The Project has had to establish some limits to accepting contributions via GitHub from people who have not yet earned write access to the project's repositories. These limits ensure that the volunteers that verify and apply the changes to make the best use of their time. Consequently, The Project is unable to accept:

- Changes too large to review on GitHub
- Typos in comments
- Changes discovered by running static analyzers over the tree (unless they include new test cases for the bugs the static analyzers found). Exceptions can be made on a case-by-case basis for "obviously correct" fixes in parts of the system that do not interact well with our testing harness.
- Changes that are theoretical, but have no specific bug or articulable behavior defect.
- Performance optimizations that aren't accompanied by before / after measurements to show improvement. Micro-optimizations are rarely worth it, as compiler and CPU technology often makes them obsolete (or even slower) in only a few years.

- Changes that are contentious. These need to be socialized on the **freebsd-arch@freebsd.org** or most appropriate mailing list first. GitHub provides a poor forum for discussing these sorts of issues.

PRs should make the Project better in some, user-visible way.

Evaluation Criteria

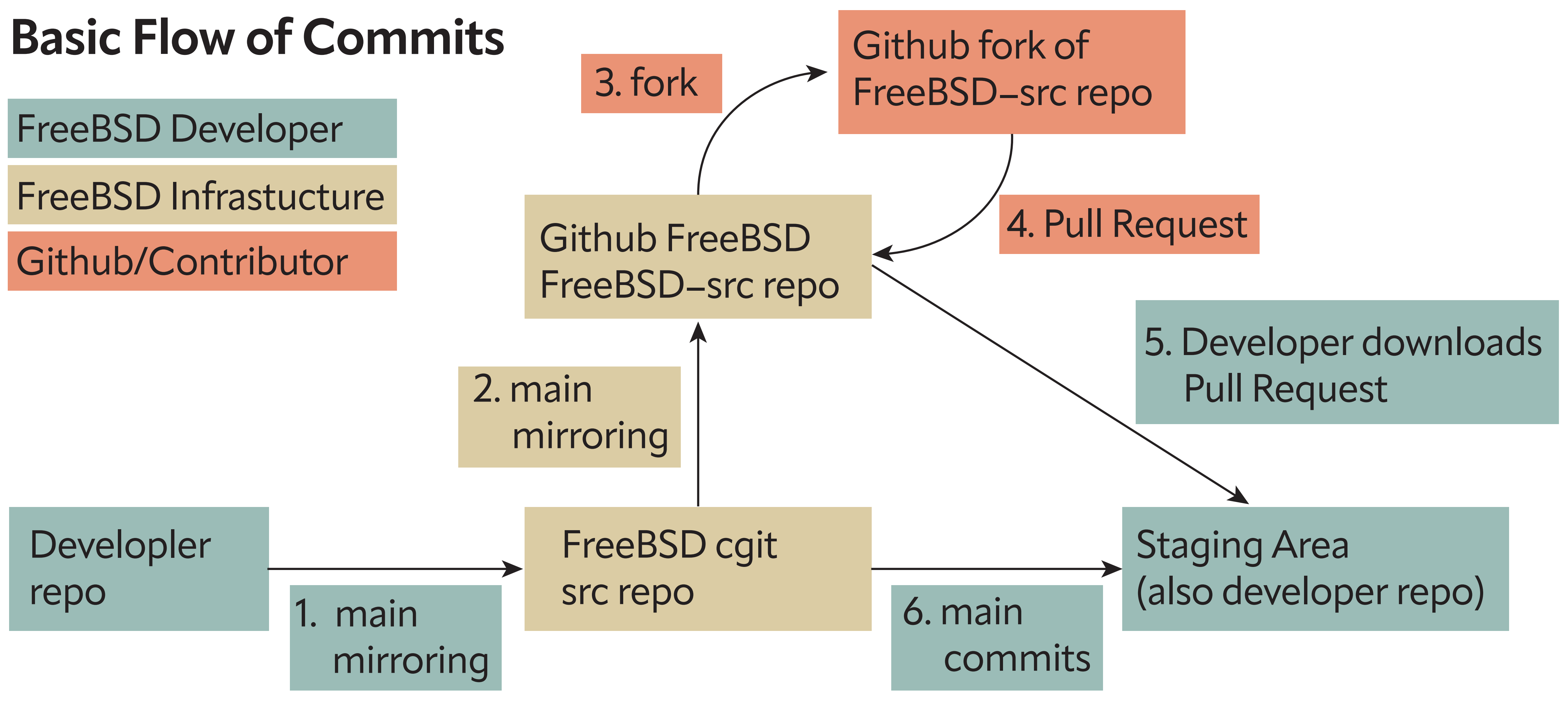
- Is the change one that the Project is accepting?
- Is the scope/scale of the change good?
  - Are there a reasonable number of commits (say less than 20)?
  - Are each of the commits a reviewable size (say less than 100 lines)?
- Does C and C++ code confirm to style(9) (or the file's current style)
- Do changes to lua confirm to style.lua(9)
- Do changes to Makefiles conform to style.Makefile(9)
- Do changes to man pages pass both `mdoc -Tlint` and `igor`?
- Have contentious changes been discussed in the proper mailing list?
- Does **make tinderbox** run successfully?
- Do the changes fix a specific, articulable problem or add a specific feature?
- Are the commit messages good?

While avoiding these pitfalls:

- Do the changes introduce new test regressions?
- Do the changes introduce behavior regressions?
- Do the changes introduce a performance regression?

Overview of the Process

At a high level, contributing to FreeBSD is a straightforward process, though getting into the details can obscure this simplicity.

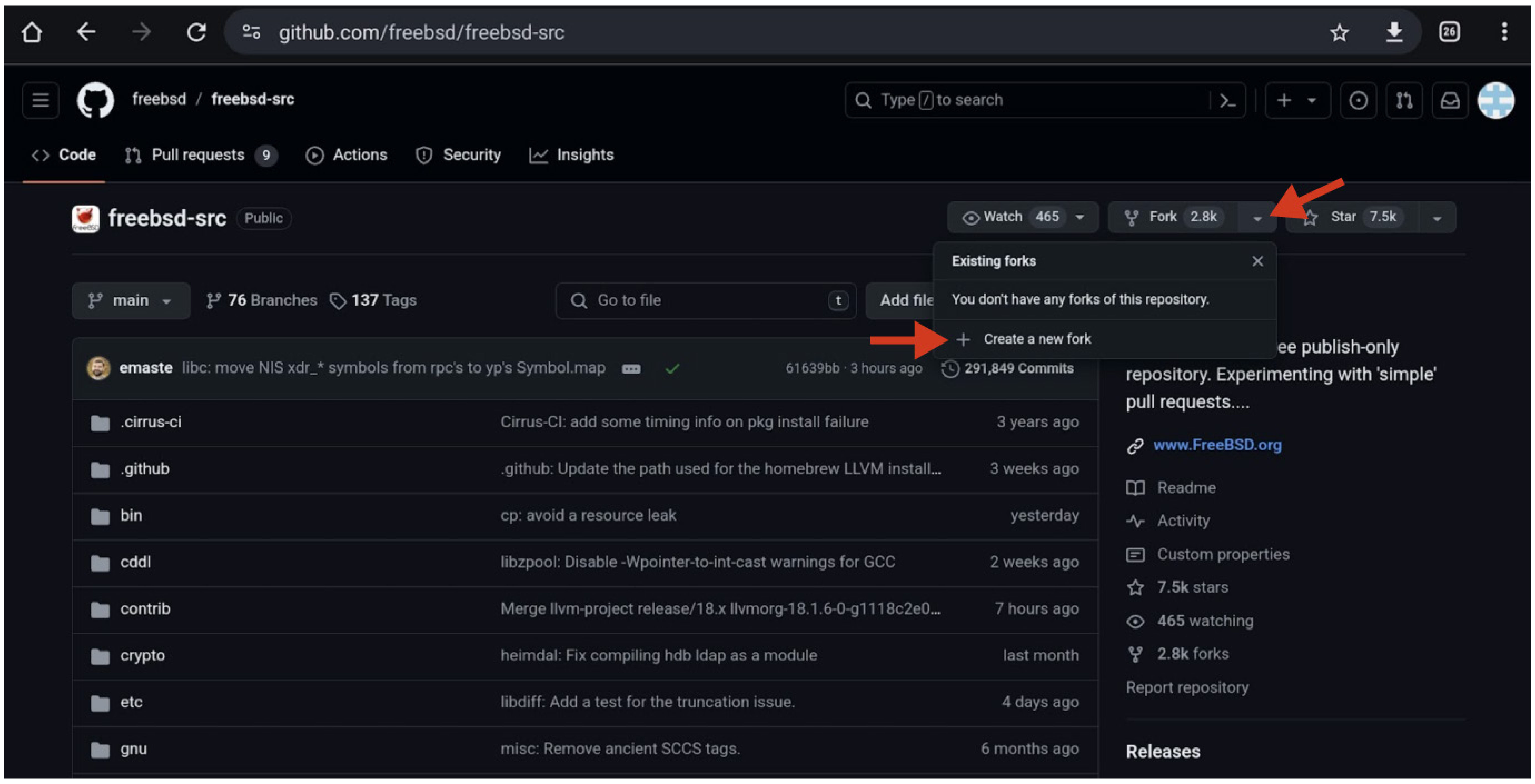


1. FreeBSD developers push commits directly to the FreeBSD repository, which is hosted in the FreeBSD.org cluster.
2. Every 10 minutes, the FreeBSD src repository is mirrored to the freebsd-src GitHub repo.
3. A user wanting to create a PR will create a branch in their fork of the freebsd-src repo
4. The changes on a user branch are used to create a FreeBSD PR.
5. A FreeBSD developer reviews the PR, provides feedback, and may request changes from the user.
6. A FreeBSD developer will push the changes into the FreeBSD src repo.

Prepping for Submitting Pull Requests

You'll need to create a GitHub account, if you don't already have one. This link will walk you through the process of creating a new GitHub account. Since many people already have a GitHub account for other reasons, we'll skip delving into the details.

The next step is forking FreeBSD's repository into your account. Using the Github web interface is the easiest way to create a fork and to explain since you will only need to do this once. Changes to your fork do not affect FreeBSD's repo. Users can fork repositories by clicking the "Fork" button as shown in **Figure 1**. You will want to click on the highlighted "Create a new fork" menu item. This will bring up a screen similar to **Figure 2**. From here, click the green "Create Fork" button.



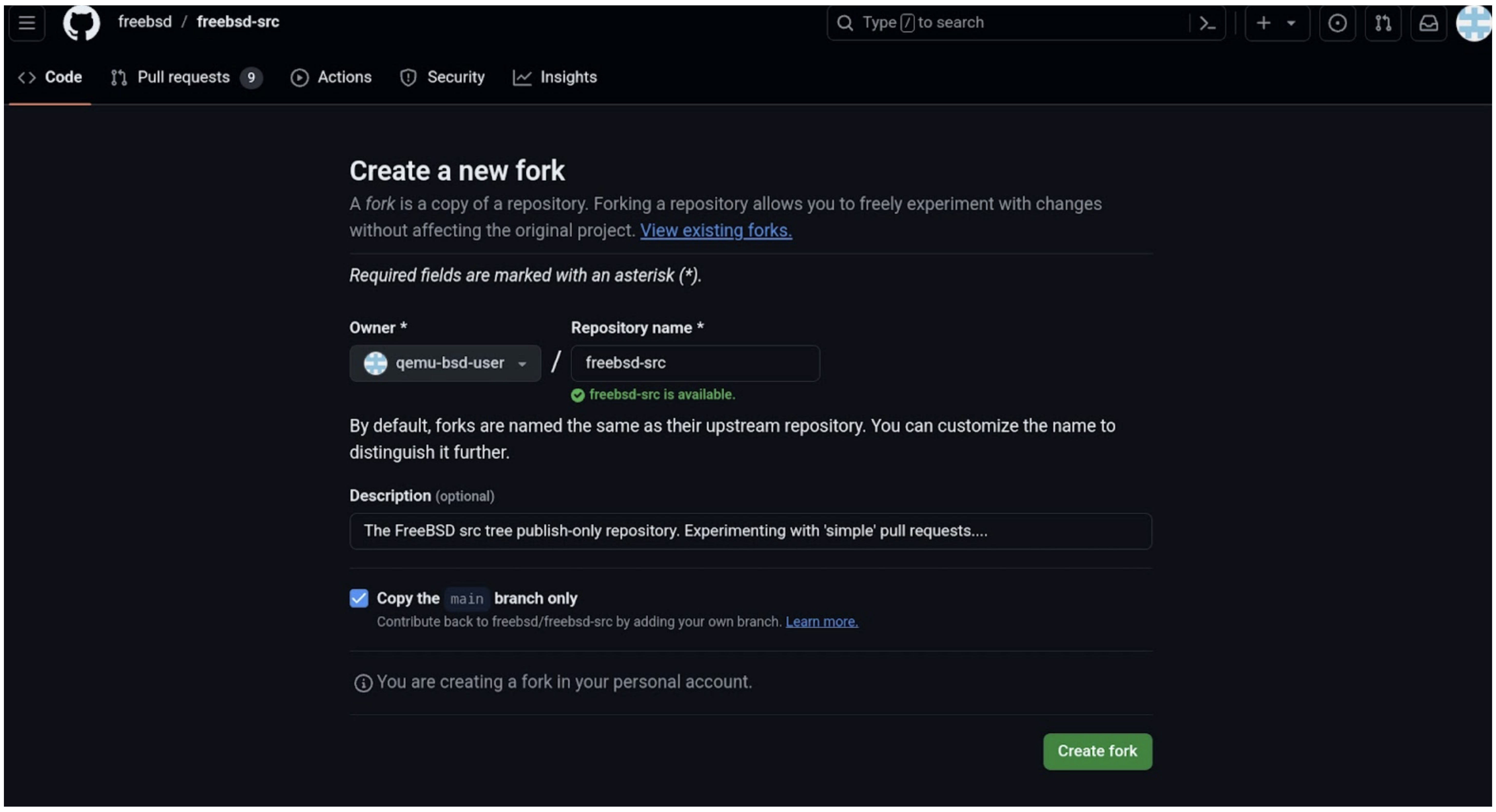
emaste libc: move NIS xdr_* symbols from rpc's to yp's Symbol.map 61639bb · 3 hours ago 291,849 Commits

.cirrus-ci	Cirrus-CI: add some timing info on pkg install failure	3 years ago
.github	.github: Update the path used for the homebrew LLVM install...	3 weeks ago
bin	cp: avoid a resource leak	yesterday
cddl	libzpool: Disable -Wpointer-to-int-cast warnings for GCC	2 weeks ago
contrib	Merge llvm-project release/18.x llvmorg-18.1.6-0-g1118c2e0...	7 hours ago
crypto	heimdal: Fix compiling hdb ldap as a module	last month
etc	libdiff: Add a test for the truncation issue.	4 days ago
gnu	misc: Remove ancient SCCS tags.	6 months ago

...ee publish-only repository. Experimenting with 'simple' pull requests....

www.FreeBSD.org

Figure 1: After clicking the down arrow next to Fork, you'll see a pop-up for the create fork dialog.



Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.

Required fields are marked with an asterisk (*).

Owner *: qemu-bsd-user / Repository name *: freebsd-src

freebsd-src is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

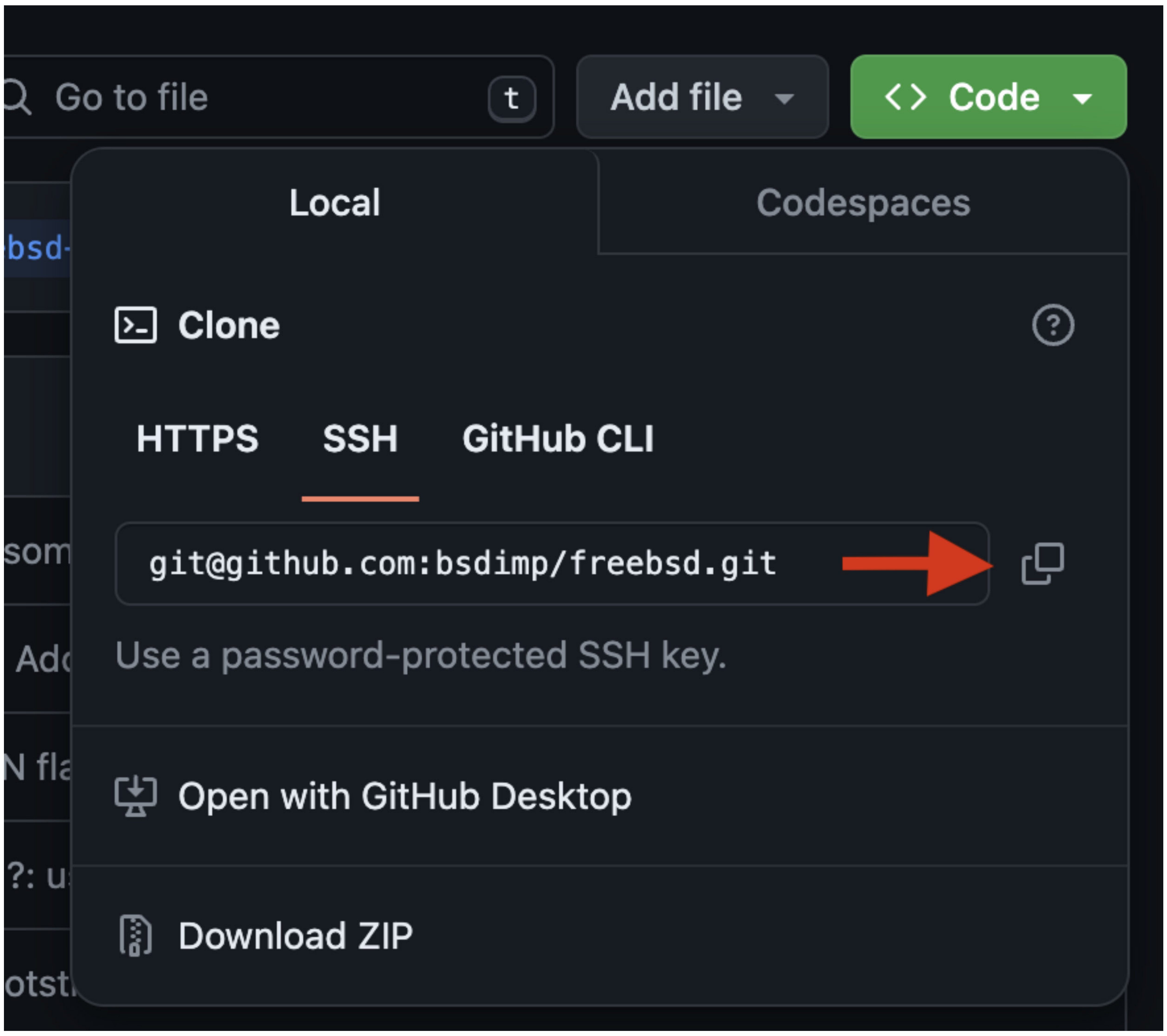
Description (optional): The FreeBSD src tree publish-only repository. Experimenting with 'simple' pull requests....

[✓] Copy the `main` branch only
Contribute back to freebsd/freebsd-src by adding your own branch. Learn more.

You are creating a fork in your personal account.

Figure 2: Creating a Fork, part 2.

Once you click on "Create Fork," the GUI will redirect to the newly forked repository. You can copy the URL you need to clone the repository in the usual spot, shown in **Figure 3**.



```
git@github.com:bsdimp/freebsd.git
```

Use a password-protected SSH key.

Figure 3: Copying the URL to clone (I forked ages ago with old repo name)

This concludes the steps you'll do with the GitHub web interface. The rest of these commands will be done in a terminal window for a host running FreeBSD. For simplicity, the screen shots have changed to the commands or the commands and the output produced by those commands.

Clone your newly created repository using the commands below.

```
% git clone
Cloning into 'freebsd-src'...
remote: Enumerating objects: 3287614, done.
remote: Counting objects: 100% (993/993), done.
remote: Compressing objects: 100% (585/585), done.
remote: Total 3287614 (delta 412), reused 815 (delta 397), pack-reused 3286621
Receiving objects: 100% (3287614/3287614), 2.44 GiB | 22.06 MiB/s, done.
Resolving deltas: 100% (2414925/2414925), done.
Updating files: 100% (100972/100972), done.
% cd freebsd-src
```

Please note you should change "user" in the above command to your GitHub username. The "-o github" will name this remote "github," which will be used in the examples below.

The PR workflow generally requires a branch. We'll assume you've followed something like the following commands, though there are many ways to use a pre-existing branch that are beyond the scope of this article.

```
% git checkout -b journal-demo
% # make changes, test them etc
% git commit
```

It is important that all the commits you make have your real name and email address as the "Author" of the commit. Git has two configuration fields for this. user.name contains your real name. And user.email has your email address. You can set them like so:

```
% git config --global user.name "Pat Bell"
% git config -global user.email "pbell@example.com"
```

In addition, please read our advice on Commit Log Messages and follow it when creating commits.

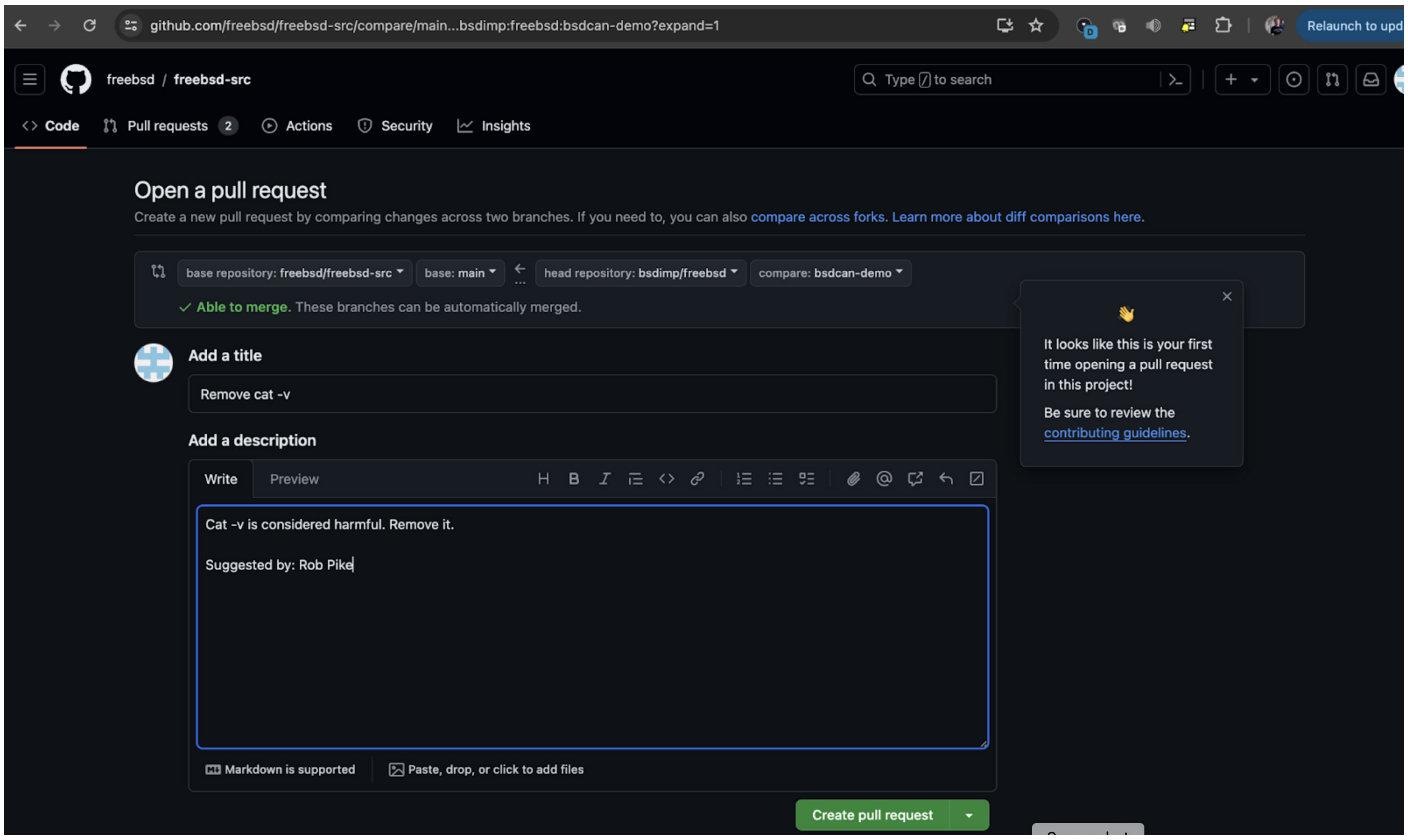
Most changes we get via PRs are small, so we'll move on to submitting them. However, if you have large changes, please read the **Evaluation Criteria** below before submitting for a smoother process.

Submitting Your Pull Request

The next step is to push the **journal-demo** branch to GitHub (as with the above, substitute your GitHub username for "user" below:

```
% git push github
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (16/16), 5.21 KiB | 1.74 MiB/s, done.
Total 16 (delta 13), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (13/13), completed with 8 local objects.
remote:
remote: Create a pull request for 'journal-demo' on GitHub by visiting:
remote:    https://github.com/user/freebsd-src/pull/new/journal-demo
remote:
To github.com:user/freebsd-src.git
 * [new branch]              journal-demo -> journal-demo
```

You'll notice that GitHub helpfully tells you how to create a pull request. When you visit the above URL, you're presented with a blank form, as shown in **Figure 4**.


github.com/freebsd/freebsd-src/compare/main...bsdimp:freebsd:bsdcan-demo?expand=1

freebsd / freebsd-src

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base repository: freebsd/freebsd-src base: main ← head repository: bsdimp/freebsd compare: bsdcan-demo

✓ Able to merge. These branches can be automatically merged.

Add a title

Remove cat -v

Add a description

Cat -v is considered harmful. Remove it.

Suggested by: Rob Pike

It looks like this is your first time opening a pull request in this project!

Be sure to review the contributing guidelines.

Figure 4: Pull request submission form.

In the "Add a Title" field, add a brief description of your work that conveys the essence of the changes. Keep this to about a dozen words, so it's easy to read. If the branch has only one commit, use the first line of the commit message for that change here. If it's multiple commits, you'll need to summarize them into one short title.

In the "Add a Description" field, write a summary of your changes. If this is a branch with just one commit, use the body of the commit message here. If there are multiple commits, then create a brief summary which briefly describes the problem solved. Explain what you changed and why, if it isn't obvious.

The example in **Figure 4** attempts to address a famous historical dispute between Bell Labs and Berkeley. It is a good example of a contentious commit outlined below. It is a good example of a contentious commit that should be socialized.

What to Expect

After your submission, the evaluation process begins. Several automated checkers will run. These make sure that the format and style of your submission conform to our guidelines. They ensure that the proposed changes compile. They will provide feedback for changes you should make before someone looks at it. Some of these tests take time, so checking back a few hours after your submission is a good idea. Items flagged by the automated testing will be among the first things our volunteers will ask you to correct, so it saves everybody time to proactively address them.

Replying to Feedback

Once you've received feedback, oftentimes code changes are required. Please make the changes that were suggested. Usually this means that you'll have to edit some subset of your changes (either commit messages, or the commits themselves). GitLab has a good tutorial on the mechanics of using **git rebase**.

Once you're made your changes, you'll need to push the changes back to your branch so the PR updates and the feedback loop starts over:

```
% git push github --force-with-lease
```

Supply Chain Attacks

Recently, a bad actor attacked the xz source base to insert code that compromised sshd on certain Linux systems. FreeBSD was unaffected by this attack due to a combination of luck and process. Our process is designed to resist such attacks by having multiple layers of protection. We review code before we allow it to be tested. We only run automated testing when it's clear there's no obvious mischief in the submissions. Questions that might seem unnecessary are often motivated by the increasingly hostile work environment with which open source projects must cope.

Wrapping up

Whether you are a casual user that has an occasional tweak to make FreeBSD better, or a more intense developer who submits so many changes that you'll earn a commit bit, the Project welcomes your submissions. This article tries to cover the basics of doing this, but is more geared to the causal user. The online resources will help for situations beyond the basics.

WARNER LOSH has been contributing to open source since before the FreeBSD project existed or the term "open source" was formally defined. He's recently been delving into the early history of Unix to discover its rich, hidden legacy. He lives in Colorado with his wife and daughter in a strawbale house heated by the sun, a small boiler, and the occasional antique computer.