# Introduction to OpenSauced


Watch this on YouTube

OpenSauced provides guidance for new contributors to find their next contribution. Our approach towards onboarding offers a way to track contributions through a GitHub-powered dashboard.

## Highlights

The Highlights feature is what we like to call the "glitzy pepperoni" of your

OpenSauced profile. It's the place where you can display your favorite open source contributions whether it's an article you wrote for Codecademy's Docs website, revising one of the lessons on freeCodeCamp's curricula, or creating a tutorial for people who use Audacity.

Eager to get started? Check outthe Effectively Highlights Your Contributions section in our free "Intro to Open Source" course to learn more.

- Ready to start contributing? Start here.

# Introduction to contributing

Contributions are always welcome, no matter how large or small. Before contributing, please read the code of conduct.

Some thoughts to help you contribute to this project

## Recommended communication style

1. Always leave screenshots for visuals changes
2. Always leave a detailed description in the Pull Request. Leave nothing ambiguous for the reviewer.
3. Always review your code first. Do this by leaving comments in your coding noting questions, or interesting things for the reviewer.
4. Always communicate. Whether it is in the issue or the pull request, keeping the lines of communication helps everyone around you.

## Setup

1. Fork one of the repositories from github/open-sauced to your own GitHub account.
2. Clone the forked repository to your local machine.
3. Run `npm ci` to install the dependencies and set up the project.

You can also use the shell commands below to get started once you have forked the repository. Make sure to replace `<your-name>` with your GitHub username.

```
git clone https://github.com/<your-name>/open-sauced
cd open-sauced
npm ci
```

# Building

```
npm run build
```

# Testing

For running the test suite, use the following command. Since the tests run in watch mode by default, some users may encounter errors about too many files being open. In this case, it may be beneficial to install watchman.

```
# the tests will run in watch mode by default
npm test
```

For more info on testing React and JavaScript, check out this course Testing JavaScript

# Applying Lint Styleguide

To check the code for formatting and linting errors, run the following command:

```
npm run lint
```

These errors will also be displayed during development, but won't prevent the code from compiling.

To fix the formatting and linting errors, run the following command instead:

```
npm run format
```

These commands use ESLint to check and fix the code.

The automated PR checks will also run these commands, and the PR will be blocked if there are any errors, so it's a good idea to run them before submitting a PR.

# Pull requests

***We actively welcome your pull requests, however linking your work to an existing issue is preferred.***

1. Fork the repo and create your branch from the default branch.
2. Name your branch something that is descriptive to the work you are doing. i.e. adds-new-thing or fixes-mobile
3. If you've added code that should be tested, add tests.
4. If you've changed APIs, update the documentation.
5. If you make visual changes, screenshots are required.
6. Ensure the test suite passes.
7. Make sure you address any lint warnings.
8. If you make the existing code better, please let us know in your PR

description.

9. A PR description and title are required. The title is required to begin with: "feat:" or "fix:"
10. Link to an issue in the project. Unsolicited code is welcomed, but an issue is required for an announcement your intentions. PR's without a linked issue will be marked invalid and closed.

*note for maintainers: All pull requests need a label to assist automation. See the template to guide which labels to use.*

## PR validation

Examples for valid PR titles:

- fix: Correct typo.
- feat: Add support for Node 12.
- refactor!: Drop support for Node 6.

*Note that since PR titles only have a single line, you have to use the ! syntax for breaking changes.*

See Conventional Commits for more examples.

*3 tips for getting your Pull Request reviewed*

You can also experiment with conventional commits by doing:

```
npm run push
```

Using the `npm run push` command is an interactive replacement for `git commit`. It enforces the conventional commits specification for writing commit messages, making it easier for developers and maintainers to understand the

changes made in a particular commit.

Assuming you are dealing with code changes and you add them using `git add`, once you are ready to commit, there are 2 ways we can proceed: `git commit` or `npm run push`. The second method is preferred, as doing a subsequent `git push` and then opening a PR would ensure the title is conforming to our standards.

### Work in progress

GitHub has support for draft pull requests, which will disable the merge button until the PR is marked as ready for merge.

## Issues

If you wish to work on an open issue, please comment on the issue with `.take` and it will be assigned to you. If an issue is not assigned, it is assumed to be open for anyone to work on. Please assign yourself to an issue before beginning work on it to avoid conflicts.

If you are contributing to the project for the first time, please consider checking the bug or good first issue labels.

In case you get stuck, please feel free to ask for help in the Discord server or GitHub Discussions.

Please note that we have a code of conduct, please follow it in all your interactions with the project and it's contributors.

# Triage team

The Triage team is inspired by expressjs/express. This team exists to create a path for making contributions to this project and open source. All Triage Team members are expected to follow this guide: TRIAGE_GUIDE.md

> **There are no minimum requirements to become a member of the Triage Team.**

For those interested in getting involved in the project or just open source in general, please request an invite to the Triage Team in this discussion.

# Funding

OpenSauced is a part of GitHub Sponsors. If you would like to contribute, please note the sponsor page for details on how funds are distributed. If you have made any contributions to the project directly or indirectly, please consider adding your profile to the FUNDING.yml.

# Community

Do you have questions? Join the conversation in our Discord.

# Coding tips

- Ask questions if you are stuck.
- Use CSS variables
- Always use rel="noreferrer" on all target="_blank" links.

# Frequently Asked Questions

1. **How do I find good first issues?** We recommend checking out sites like Good First Issues and First Timers Only. They will lead you to projects and issues worth working on. We also recommend using the `Good First Issue` label in the search engine of the project you want to contribute to. Check out this YouTube Short from GitHub to learn more. Furthermore, check out our daily "Who's looking for open source contributors?" on Dev.to

2. **My contribution does not show up on my OpenSauced profile. How do I fix it?** We suggest creating a Insight page or adding your merged pull request to the Highlights Page.

3. **I want to find "X" language or frameworks data. How do I find it?** Post your suggestion in our Feedback repository. We'd love to see it! 😄

# License

# Code of Conduct

## Our pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

## Our standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks

- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

# Our responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

# Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

# Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at hello@briandouglas.me. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

# Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://contributor-covenant.org/version/1/4

# Triage guide

## How do I join the triage team?

1. Sign up for opensauced.pizza
2. Leave a reply in the invite discussion.

## Issue triage process

When a new issue or pull request is opened the issue will be labeled with `needs triage`. If a triage team member is available they can help make sure all the required information is provided. Depending on the issue or PR there are several next labels they can add for further classification:

- `needs triage`: This can be kept if the triager is unsure which next steps to take
- `awaiting more info`: If more info has been requested from the author, apply this label.
- `question`: User questions that do not appear to be bugs or enhancements.
- `discuss`: Topics for discussion. Might end in an `enhancement` or `question` label.
- `bug`: Issues that present a reasonable conviction there is a reproducible bug.
- `enhancement`: Issues that are found to be a reasonable candidate feature additions.

In all cases, issues may be closed by maintainers if they don't receive a timely

response when further information is sought, or when additional questions are asked.

# Approaches and best practices for getting into triage contributions

Review the project's contribution guideline if present. In a nutshell, commit to the community's standards and values. Review the documentation, for most of the projects it is just the README.md, and make sure you understand the key APIs, semantics, configurations, and use cases.

It might be helpful to write your own test apps to re-affirm your understanding of the key functions. This may identify some gaps in documentation, record those as they might be good PR's to open. Skim through the issue backlog; identify low hanging issues and mostly new ones. From those, attempt to recreate issues based on the OP description and ask questions if required. No question is a bad question!

# Labeling good first issues

Issues labeled as `good first issue` represent a curated list of easy contributions for new contributors. These issues are meant to help folks make their first contribution to open-source and should not require an excessive amount of research or triaging on the contributor's part.

*All good first issues should include one or more of the following: a solution, a suggestion for a solution, links to components, or in which issue occurs.*

- Issues that `needs triage` cannot be labeled as `good first issues`.
- It is better to have no `good first issue` labeled issues than to have a

`good first issue` confusing enough to deter a contributor from contributing.

# Removal of triage role

There are a few cases where members can be removed as triagers:

- Breaking the CoC or project contributor guidelines
- Abuse or misuse of the role as deemed by the TC
- Lack of participation for more than 6 months

If any of these happen we will discuss as a part of the triage portion of the regular TC meetings. If you have questions feel free to reach out to any of the TC members.

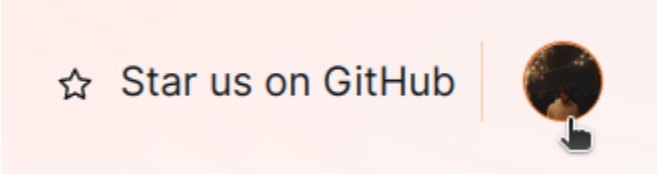
# Other helpful hints:

- When reviewing the list of open issues there are some common types and suggested actions:
  - New/unattended issues or simple questions: A good place to start
  - Hard bugs & ongoing discussions: always feel free to chime in and help
  - Issues that imply gaps in documentation: open PRs with changes or help the user to do so
- For recurring issues, it is helpful to create functional examples to demonstrate (publish as gists or a repo)
- Review and identify the maintainers. If necessary, at-mention one or more of them if you are unsure what to do
- Make sure all your interactions are professional, welcoming and respectful to the parties involved.

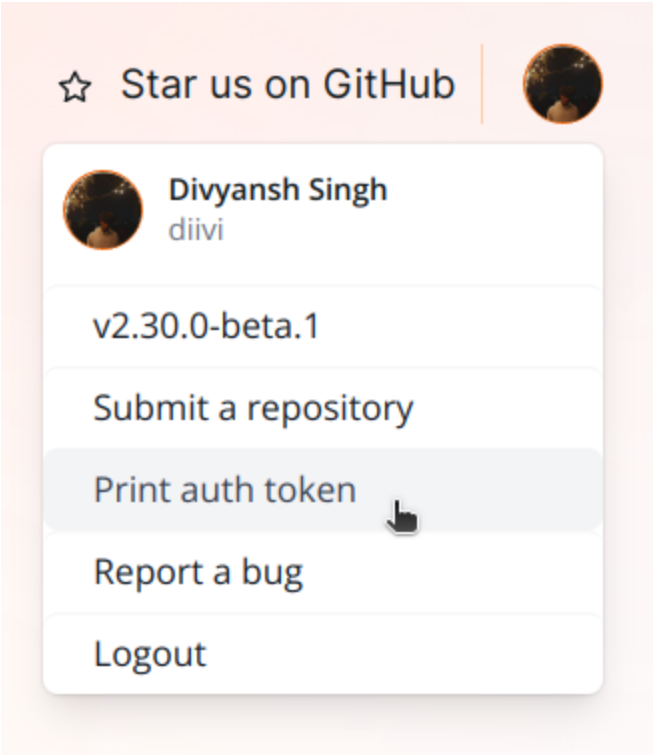
# Set up Authentication

To interact with the OpenSauced public API as an authenticated user, you need to obtain an authentication token. The following steps outline how to obtain an authentication token from the hot.opensauced.pizza website:

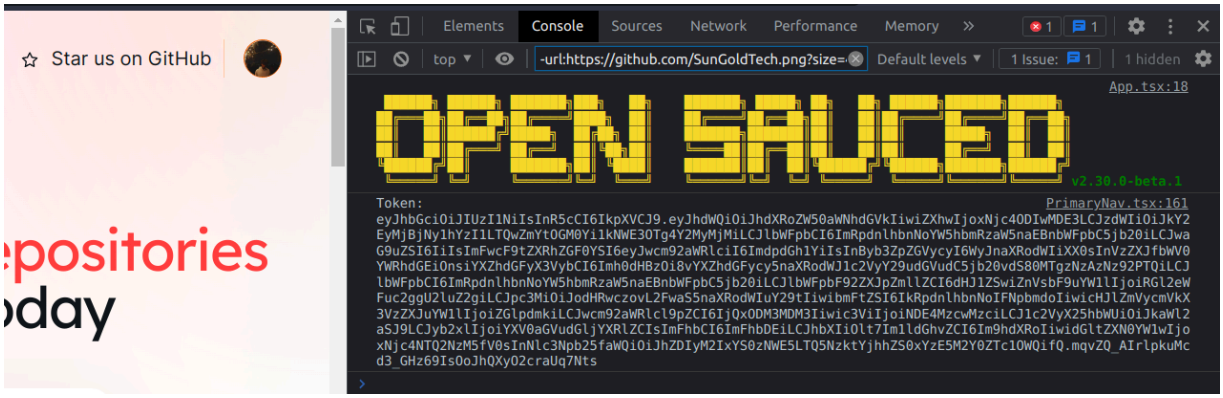
1. Click on your avatar in the top right corner of the page. This will open a dropdown menu.




2. Click on the **Print Auth Token** option. Don't worry, this is a safe operation. The token is only printed to the console.




3. Copy the token that is printed to the console.




4. You can now use this token to make authenticated requests to the OpenSauced public API by including it in the Authorization header of your requests. For example:

```
const response = await fetch("https://api.opensauced.pizza/v1/
```

# Introduction to storybook

Storybook is being leveraged to mock out visual React components. The latest version of the design system can be found at this URL.

To run storybook, use this command:

```
npm run storybook
```

## UI categories

Storybook is broken into several categories:

- **Button:** These are the button elements that appear in the project in various forms. They primarily are the Button component in the project but can also be icons.
- **Cards:** These are the main container elements in the project. Each item represents a live component in their current form in the project.
- **Primitives: These are the basic styling of base HTML components.**
- **Nav:** This is the main navigation bar for the project. There are two states, when there is no user logged in and when a user is logged in.
- **Footer:** This represents the various footers for the project.
- **Homepage:** This is the main component for the project homepage and shows the home page in its current form.

- **Miscellaneous:** These are components that currently don't fit neatly into the above categories.

# Making changes to storybook

This section details how to make changes to Storybook, mainly creating new categories or UI elements.

# Adding a new category

To add a new category, a new file needs to be added to `/stories`. Please follow the naming convention of `*Previous File Number + 1*-*Name of Story Capitalized*-stories.js` when creating a new file. In the file ensure you have this code in the file:

```
export default {
    title: "*Name of category*"
};
```

# Adding a new UI element

To add a new UI element to to an existing category, add the following code to that category's file:

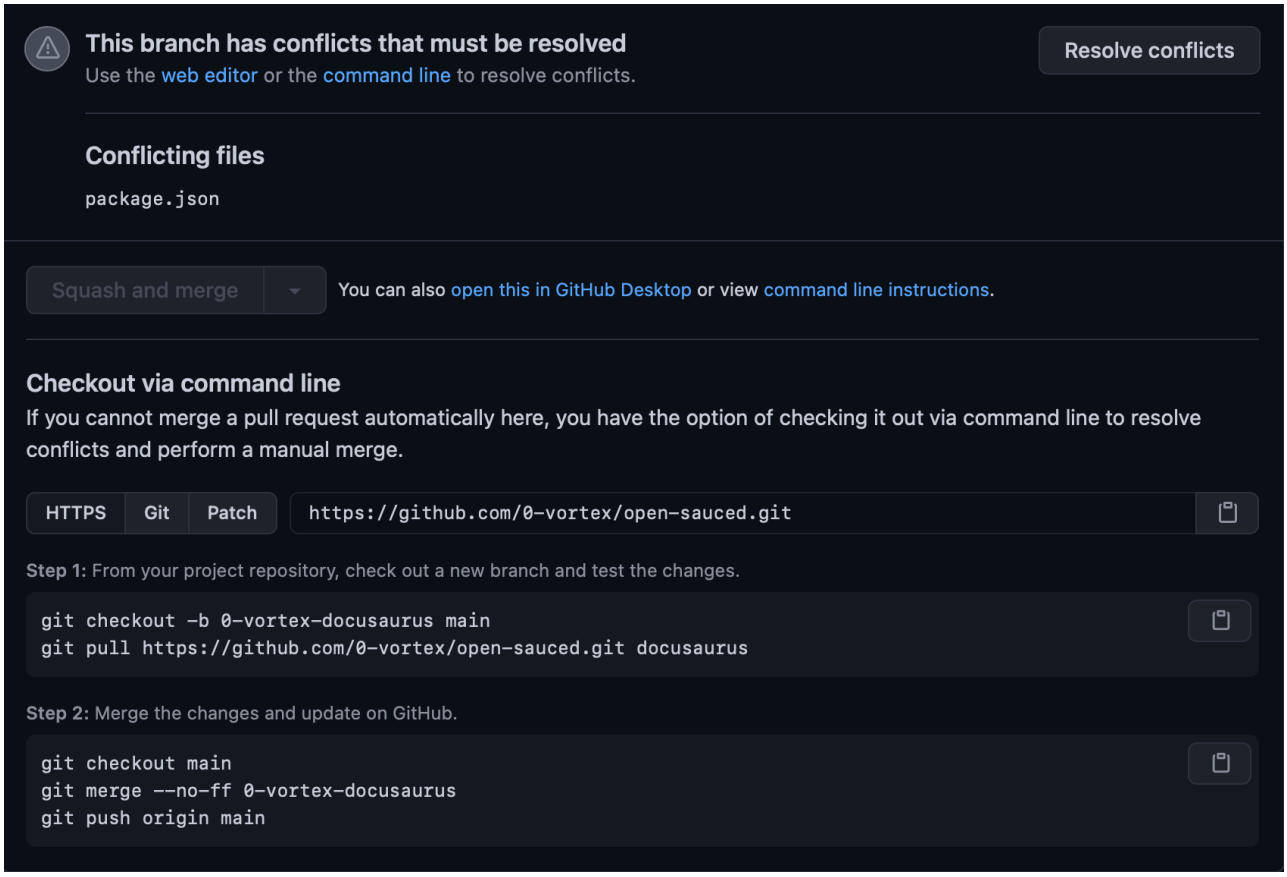
```
export const *Name of UI Element* = () => (

);
```

# Resolve merge conflicts

Pretty often when opening a pull request it is very likely to run into merge conflicts as the release process is generally updating `npm-shriknwrap.json`.

To better illustrate the commands listed here at will use commits and screenshots from open-sauced#1078.

In literally every case it is advised ***not*** to use the `Resolve conflicts` button as follows:




The above will at best achieve a ready to merge pull request with visible inconsistencies.

# Repository setup

Fork and clone the project using the `gh` command line interface:

```
gh repo clone 0-vortex/open-sauced
```

Running `git remote -v` will output:

```
origin git@github.com:0-vortex/open-sauced.git (fetch)
```

Fork and clone the project using the `git` command line interface:

```
git clone git@github.com:0-vortex/open-sauced.git
```

Running `git remote -v` will output:

```
origin git@github.com:0-vortex/open-sauced.git (fetch)
origin git@github.com:0-vortex/open-sauced.git (push)
```

As an additional step for this tutorial we need to add the `upstream` remote:

```
git remote add upstream git@github.com:open-sauced/open-sauced.git
```

# Update

First get the default branch changes:

```
git fetch origin --recurse-submodules=no --progress --prune
git checkout main --
git fetch upstream --recurse-submodules=no --progress --prune
git merge upstream/main --no-stat -v
```

# Merge with upstream

Then merge with the forked up-to-date `beta` (default branch):

```
git merge origin/main --no-ff -v
```

You will see something similar to:

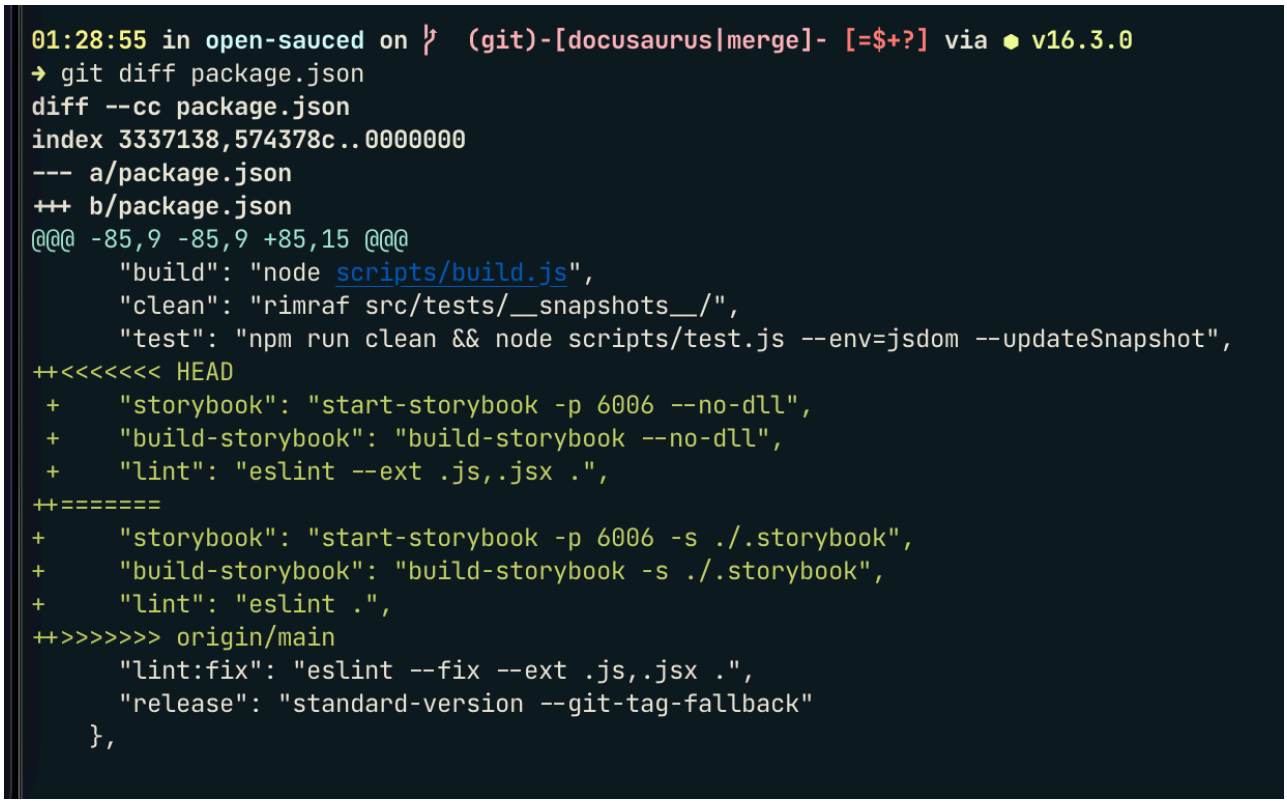
```
01:22:44 in open-sauced on  docusaurus [$?] is 📦 v0.21.21 via ⬢ v16.3.0
➜ git merge origin/main --no-ff -v
Auto-merging package.json
CONFLICT (content): Merge conflict in package.json
Automatic merge failed; fix conflicts and then commit the result.

01:22:50 in open-sauced on  (git)-[docusaurus|merge]- [=$+?] via ⬢ v16.3.0
x1 ➜ git status
On branch docusaurus
Your branch is up to date with 'origin/docusaurus'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
        modified:   .babelrc
        new file:   .storybook/index.css
        modified:   .storybook/main.js
        new file:   .storybook/preview-head.html
        new file:   .storybook/preview.js
        modified:   CHANGELOG.md
        modified:   config/polyfills.js
        modified:   npm-shrinkwrap.json
        modified:   src/styles/Background.js

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   package.json
```

# Review changes

To see what the changes are do a:

```
git diff package.json
```

It will look like this:




# Resolve conflicts

Since this pull request does not modify the `package.json` file it is safe to fast forward the changes from `origin/main`:

```
# overwrite with origin/main changes
git show :3:package.json > package.json
```

A more traditional way of doing the same thing is:

```
# make a local copy of all changes and use --theirs
# --theirs strategy overwrite with origin/main changes
git show :1:package.json > base.package.json
git show :2:package.json > branch.package.json
```

# Commit changes

Not making any assumptions about editor preferences running this will open the configured editor with a default commit message:

```
git commit
```

That should look like this:

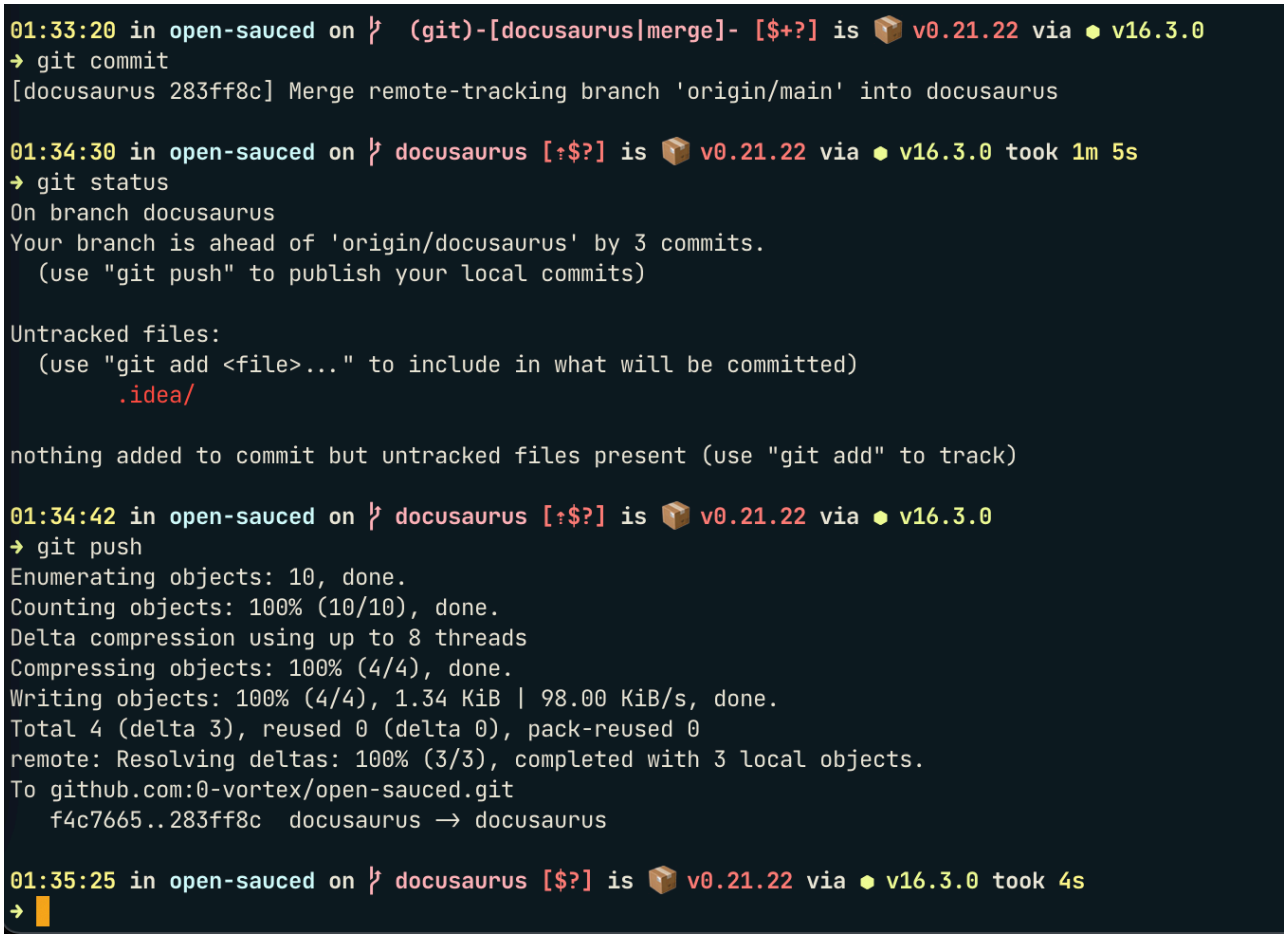
```
Merge remote-tracking branch 'origin/main' into docusaurus

# By Matthew (1) and bdougie (1)
# Via bdougie
* origin/main:
  chore(release): 0.21.22
  feat: storybook dark mode (#1061)

# Conflicts:
#	package.json
#
# It looks like you may be committing a merge.
# If this is not correct, please run
#	git update-ref -d MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch docusaurus
# Your branch is up to date with 'origin/docusaurus'.
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#	modified:   .babelrc
#	new file:   .storybook/index.css
#	modified:   .storybook/main.js
#	new file:   .storybook/preview-head.html
#	new file:   .storybook/preview.js
#	modified:   CHANGELOG.md
#	modified:   config/polyfills.js
#	modified:   npm-shrinkwrap.json
#	modified:   package.json
#	modified:   src/styles/Background.js
#
# Untracked files:
#	.idea/
#

```

# Push updated pull request

One more security check to make sure your branch has not diverged and push:

```
git status
git push
```

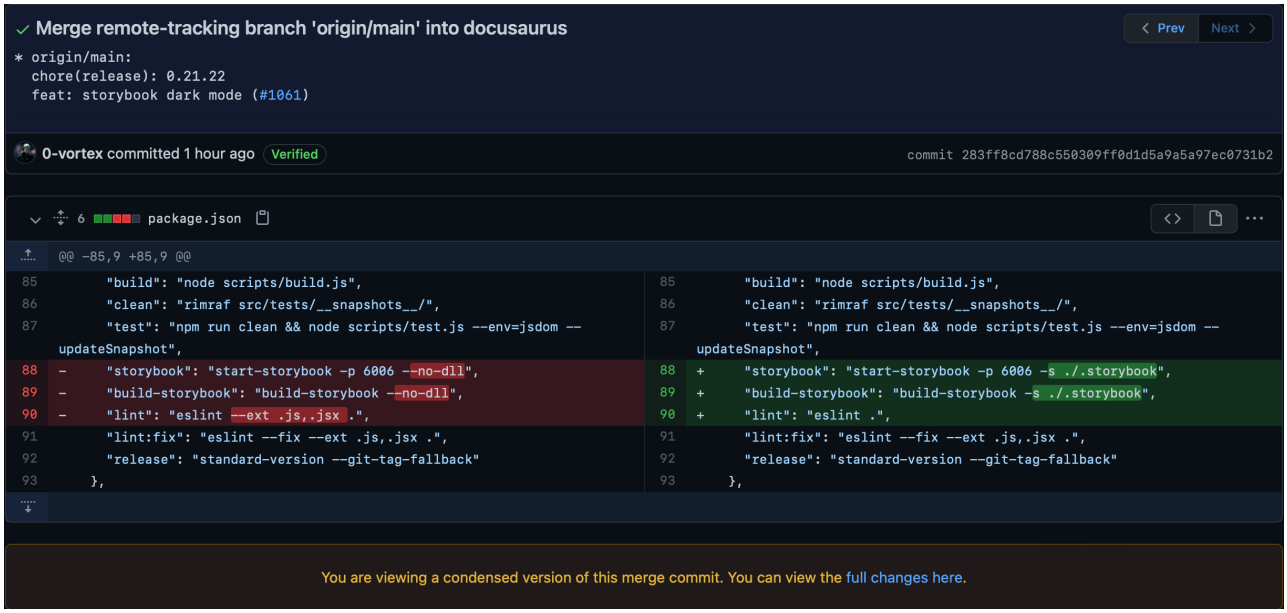
It should look something like this:



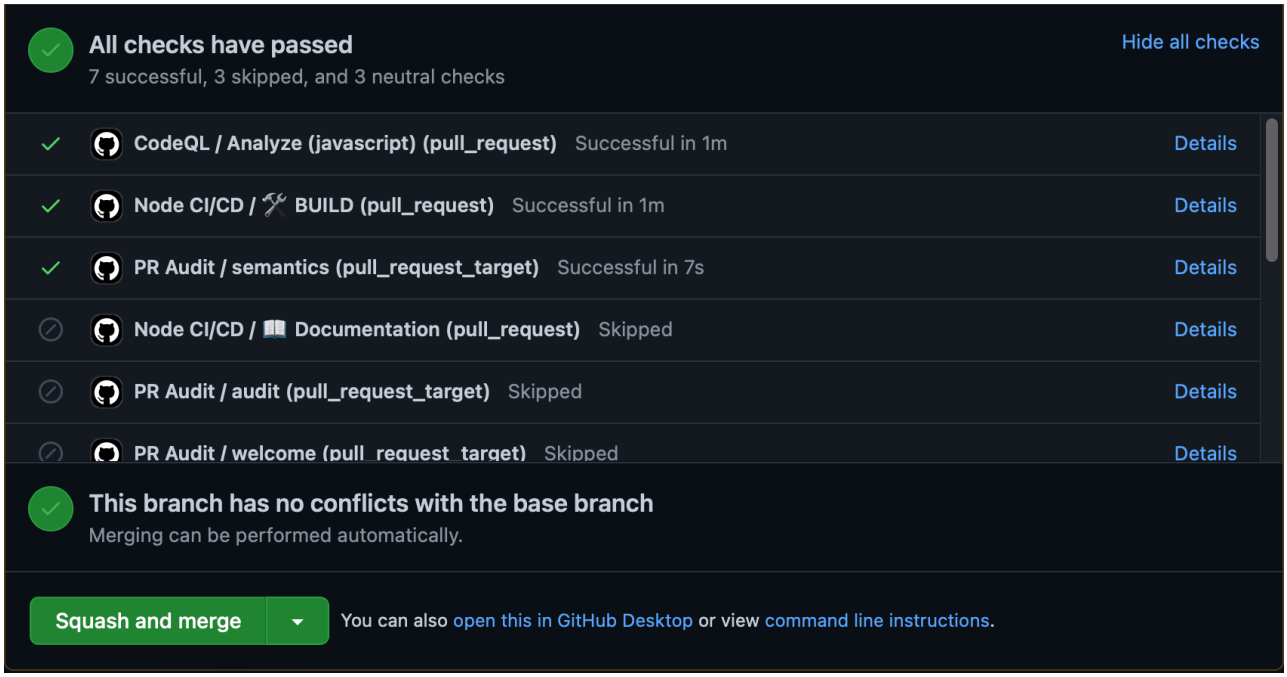
# Review your pull request

The result of the above commands can be viewed at
283ff8cd788c550309ff0d1d5a9a5a97ec0731b2

GitHub will conveniently display only you merge conflict changes:




And it's ready to merge:




# Dependency updates

When dealing with dependency and lock file updates there are multiple use cases to consider, however as a baseline, the OpenSauced triage team will not prioritize parallel main features as seen in the roadmap.

However when that happens, it is advised to:

- fast-forward `npm-shrinkwrap.json`
- fast-forward deleted and modified `upstream/beta` changes to `package.json`
- fast-forward your added lines to `package.json`
- run `npm ci` to delete local modules and create dependency resolution from `upstream/beta`

Visual diffing is advised however not following the git commit history procedure will result in a rogue pull request that scope creeps into dependency updates.

Generally speaking, just adding things to a lockfile will not be troublesome and since this is a licensed project, we should be careful when adding

dependencies.

# Setting up a new repository

## How do I Join the Maintainers Team?

1. Sign up for opensauced.pizza
2. Join discord.
3. Prove your pizza worth!

## Requirements

For the purpose of this tutorial, our target demo repository will be called `open-sauced/npx-check-engines`.

The steps described here mirror open-sauced/check-engines.

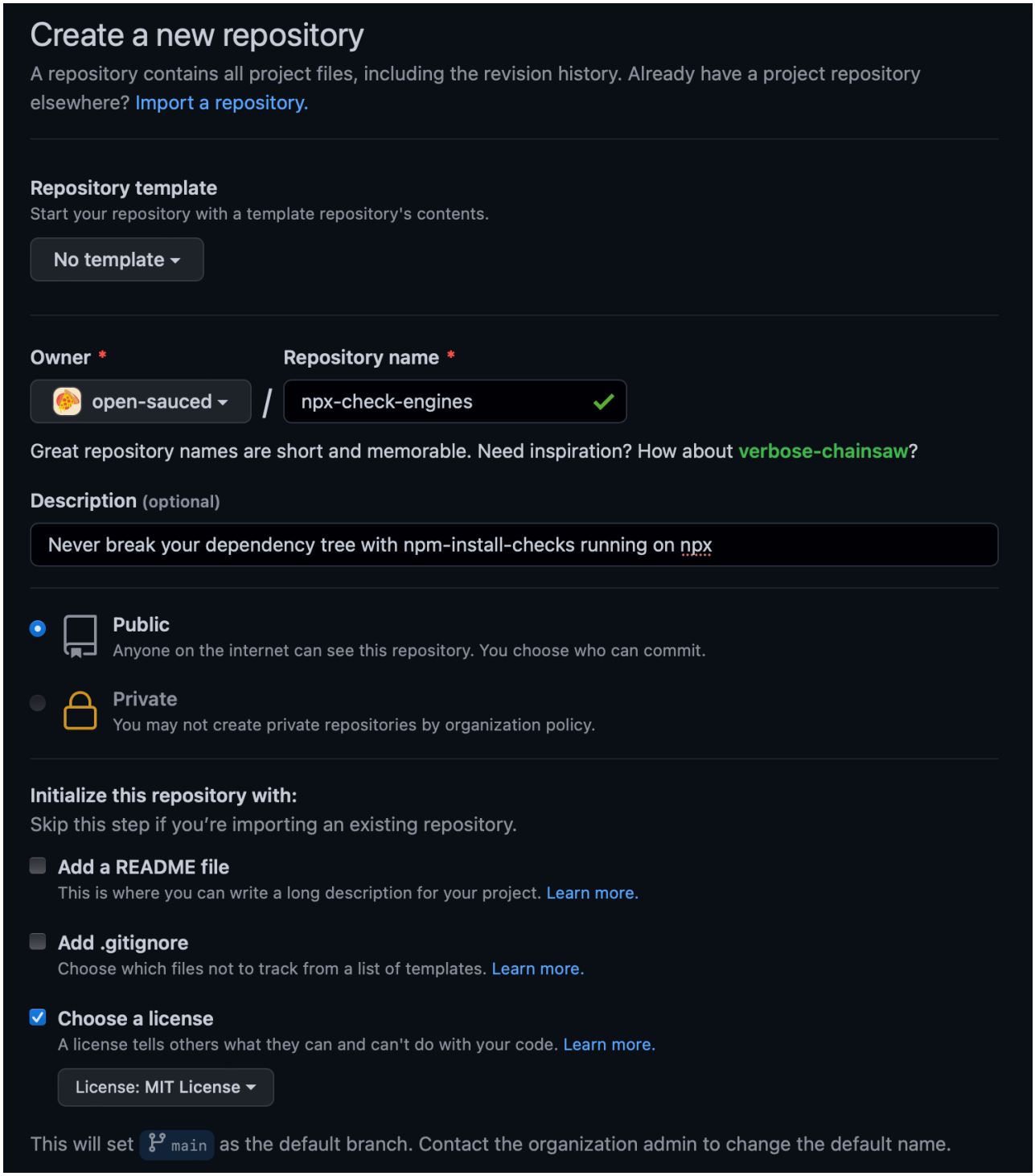
The octoherd scripts assume you have exported a programatic token similar to:

```
export GH_TOKEN="ghp_Q8TZZT9ypgqw3EeABoCWPcwZBHpjZJ9hI42n"
```

## Creating a new repo

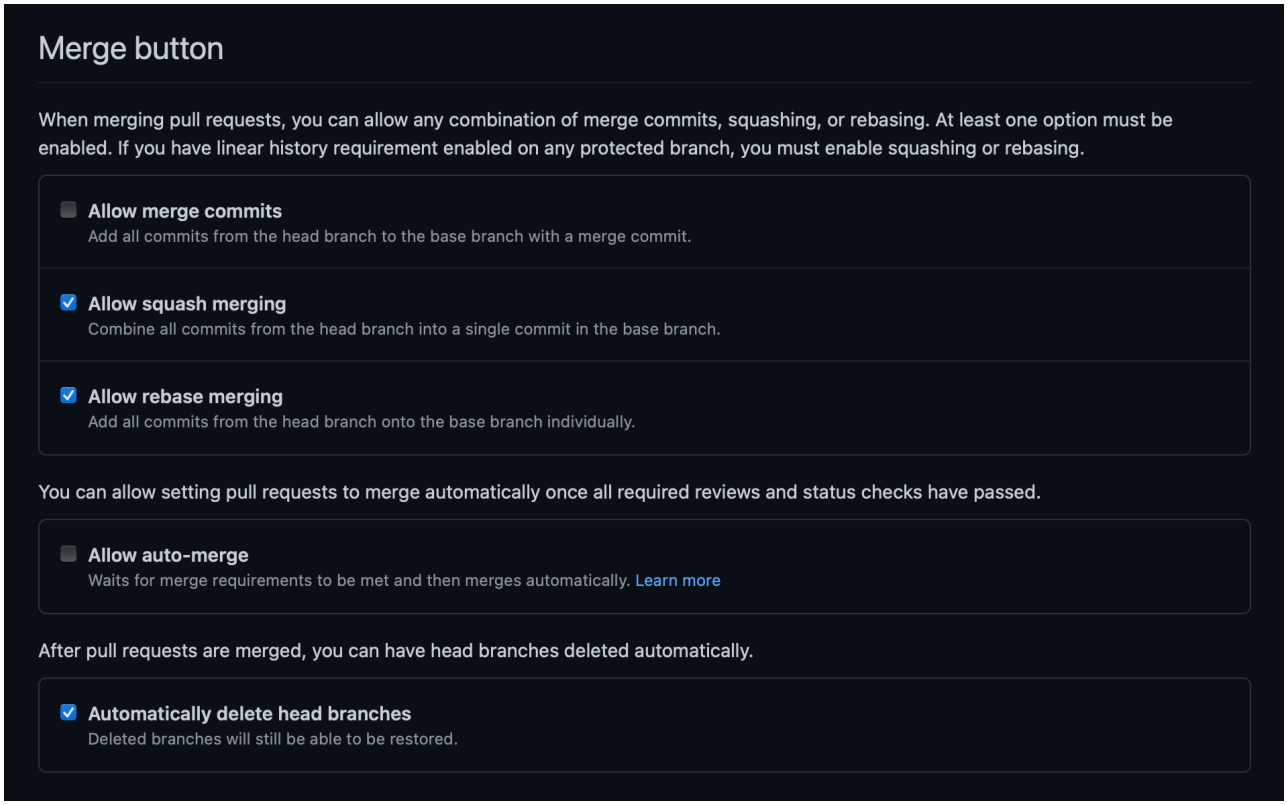
Don't spend too much time thinking of a name or a catchy description, just set

license to MIT and rocket jump!



# Syncing settings with opensauced.pizza

Squashing pull requests is the minimum requirement but the other options are quite useful at various stages of development.




Copy most of the relevant settings with:

```
npx octoherd-script-sync-repo-settings \
  --template "open-sauced/open-sauced" \
  -T $GH_TOKEN \
  -R "open-sauced/check-engines"
```

Otherwise you can disable "Projects" and "Wikis" for the selected repository as we are handling them on a larger scale.

# Syncing labels with opensauced.pizza

The default labels have some missing emojis. Copy the rest with:

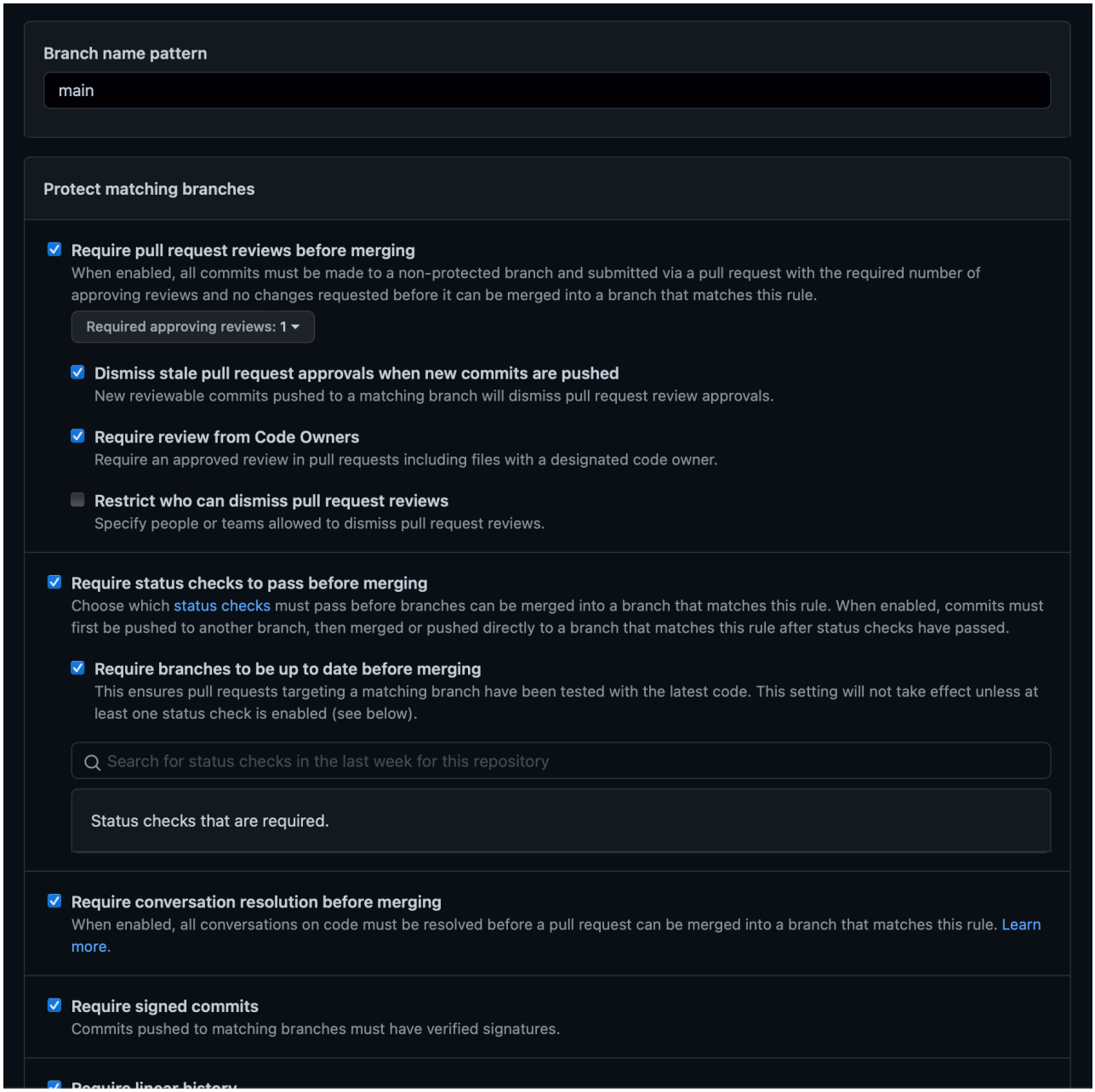
```
npx octoherd-script-copy-labels \
  --template "open-sauced/open-sauced" \
  -T $GH_TOKEN \
  -R "open-sauced/check-engines"
```

Then go back to your repository and delete:

- documentation
- 👀 needs-triage (green background one)
- other potential duplicates if the above race condition is different

# Syncing branch protections with opensauced.pizza

This topic is more complex but in a sense tap the main branch and enable everything except "Restrict who can dismiss pull request reviews" and "Restrict who can push to matching branches" in the first section.




The "Rules applied to everyone including administrators" is more on an unused override.

Most of the time this process is super manual but in the limited cases where we need this run:

```
npx @octoherd/script-sync-branch-protections \
  --template "open-sauced/open-sauced" \
  -T $GH_TOKEN \
  -R "open-sauced/check-engines"
```

# Setting up workflows

Most collaborative projects require compliance flows powered by amannn/action-semantic-pull-request and actions/first-interaction.

Pull requests require triage powered by bdougie/take-action.

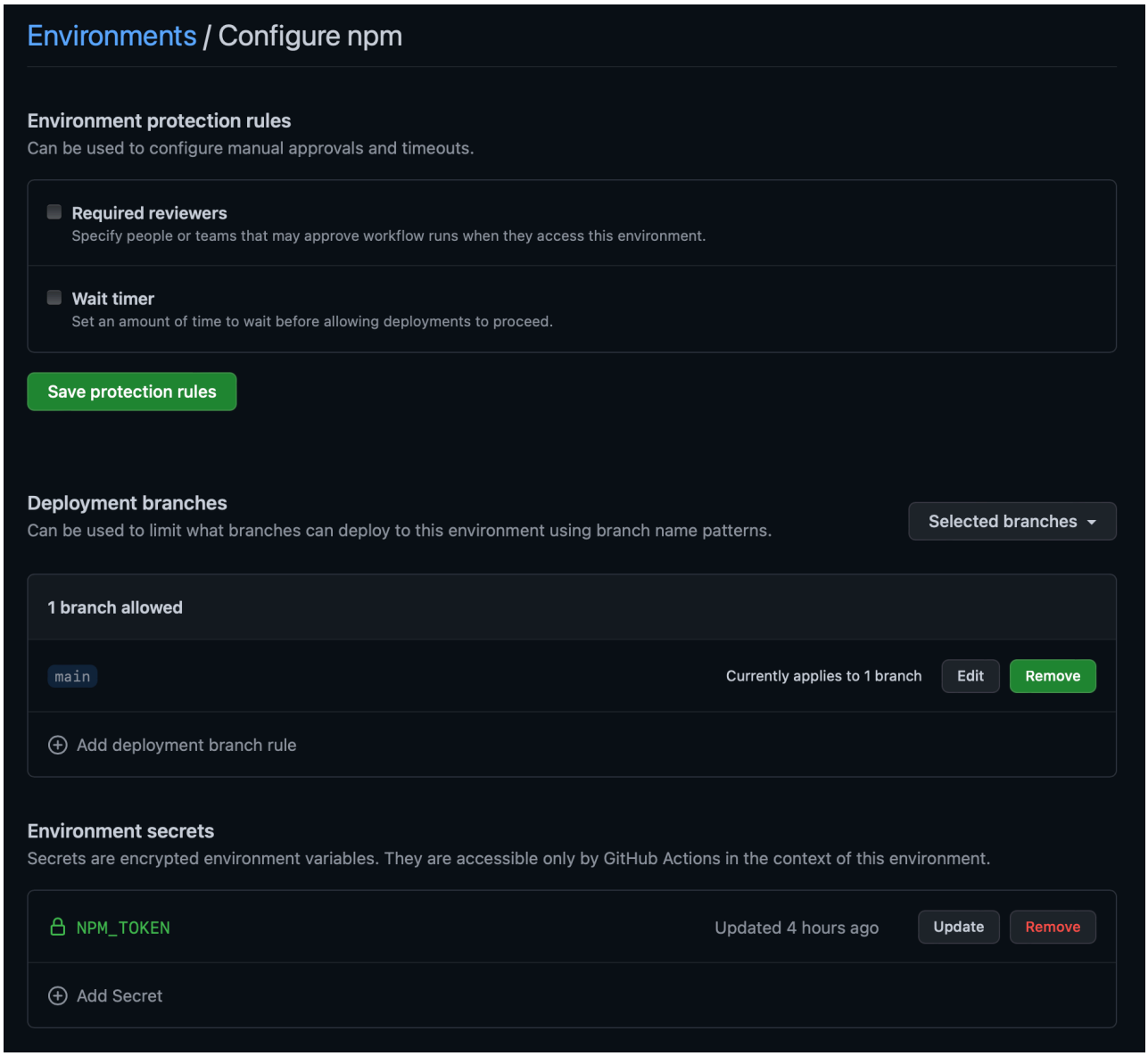
Most `node` projects will require release automation powered by @open-sauced/semantic-release-conventional-config.

Other development workflows are less common and opinionated towards decentralised collaboration. Use these as example backbones for your new repository.

# Setting up environments and secrets

As you may have noticed in the previous step or in the actions visualisations, the release workflows enable named environments.

These have to be manually set up, along with their secrets and branch protections.




If using `npm` or `ghcr` it is likely you will add a couple variables here.

# @open-sauced/check-engines

## Description

The `npm` package `@open-sauced/check-engines` is designed to help contributors install dependencies conforming to the `engines` property in package.json.

## Dependencies

This package uses the following modules:

- npm-install-checks

## Installation

```
npm install --save-dev @open-sauced/check-engines
```

Add the verification scripts to your `scripts` section in the `package.json` file:

```
{
  "scripts": {
    "preinstall": "npx @open-sauced/check-engines"
```

The reason why we provide `npx` in the `scripts` section is for the people using this as a development enhancement, interactive configurations or trimmed dependency trees, where using `npx` is preferred over installing all the dependencies at once.

# Usage

Use your favourite package manager to install dependencies in your project or, if you set it as global verification system:

```
{
  "scripts": {
    "check-engines": "npx @open-sauced/check-engines"
    "preinstall": "npm run check-engines",
    "prestart": "npm run check-engines"
  }
}
```

# Advanced usage

If you have an API or any other non-library type of application, you can decouple this package from any install scripts and just use it as a verification:

```
{
  "scripts": {
    "check-engines": "npx @open-sauced/check-engines"
    "prestart": "npm run check-engines"
  }
}
```

A more traditional approach not using `pre` or `post` scripts, this example enables the check only for local machine development:

```
{
  "scripts": {
    "check-engines": "npx @open-sauced/check-engines"
    "start": "...",
    "dev": "npm run check-engines && npm start -- --watch"
  }
}
```

# FAQ

## Usage on older `node` and `npm` versions

Older `node` and `npm` versions won't be able to run this package, depending on versions the scripts section could be ignored completely.

If you have that use case, this package is only worth enabling for progressive contributors frequently missing the legacy support of the respective module and forcefully upgrading dependencies - them running newer versions will force the error message and explicitly disable.

## Why not use `check-engines` or `engine-strict`

As described in the `npm@6` engine-strict docs:

> Prior to npm 3.0.0, this feature was used to treat this package as if the user had set engine-strict. It is no longer used.

In `npm@6` and later this was re-introduced as a config flag.

This package is designed with multiple legacy use cases in mind.

## Library usage

If you are using this module in a library package, be advised that any `*install` script will run in the parent module when installed.

For example, given a module `@demo-org/demo-package` with a `preinstall: "npx @open-sauced/check-engines"` script, running `npm install @demo-org/demo-package` will require your locally installed `node` and `npm` versions to match the `engines` section of your `package.json` - if that is not set, nothing should happen and this package is a stray dependency in either `@demo-org/demo-package` or the module you are running this command in.

# Contributing

We're always happy to onboard people into open source!

Check out the repository at @open-sauced/check-engines ❤

# @open-sauced/conventional-commit

## Description

The `npm` package `@open-sauced/conventional-commit` is designed to help users `git commit` using commitizen and conventional commits.

## Dependencies

This package uses the following modules:

- cz-cli
- cz-conventional-changelog

## Installation

```
npm install --save-dev @open-sauced/conventional-commit
```

Add the verification scripts to your `scripts` section in the `package.json` file:

```
{
  "scripts": {
    "push": "npx @open-sauced/conventional-commit"
```

The reason why we provide `npx` in the `scripts` section is for the people using this as a development enhancement, interactive configurations or trimmed dependency trees, where using `npx` is preferred over installing all the dependencies at once.

# Usage

All you have to do is run the script next to your `package.json`:

```
npx @open-sauced/conventional-commit
# or
npx conventional-commit
```

# Advanced usage

The most common use case for this package is to run it instead of the `git commit` command inside your `npm` scripts:

```
{
  "scripts": {
    "push": "npx @open-sauced/conventional-commit"
  }
}
```

or

```
{
  "scripts": {
    "push": "npx conventional-commit"
```

If you want to ensure local-only usage:

```
{
  "scripts": {
    "push": "conventional-commit"
  }
}
```

# FAQ

# Contributing

We're always happy to onboard people into open source!

Check out the repository at @open-sauced/conventional-commit ❤

# @open-sauced/semantic-release-conventional-config

## Description

The `npm` package `@open-sauced/semantic-release-conventional-config` is designed to help `npm` packages auto-release to `npm` or `ghcr` registries while generating github releases and changelog using conventional commit convention.

Version 2 supports alpha and beta pre-releases using coresponding branches.

## Dependencies

This package uses the following modules:

- `@semantic-release/commit-analyzer`
- `@semantic-release/release-notes-generator`
- `conventional-changelog-conventionalcommits`
- `@semantic-release/changelog`
- `@semantic-release/npm`
- `@google/semantic-release-replace-plugin`
- `semantic-release-license`

- `@semantic-release/git`
- `@semantic-release/github`
- `@eclass/semantic-release-docker`
- `@semantic-release/exec`
- `execa`

# 🖥️ Requirements

Most important limitations are:

- `GITHUB_TOKEN` for everything
- `NPM_TOKEN` for public `npm` library
- `docker` containers need to be built beforehand

You can skip here if you are using elevated Private Access Token, however we don't recommend going down that path.

No force push or admin cherries branch protections for the following branches:

- `main` - required
- `alpha` - optional, pre-release branch
- `beta` - optional, pre-release branch
- `next` - optional, next channel
- `next-major` - optional, next major
- `vX[.X.X]` - maintenance releases

If you use more than the main branch, optionally create an environment that is limiting where pushes can come from and enable the merge strategy.

We are using `production` in our examples, if you copy paste them you will find

this new environment generated in your settings! 🍕

## 🧪 GitHub actions usage

Since version 3 it is possible to use semantic-release without any trace of it or the open-sauced configuration anywhere in the dependency tree.

Docker containers are pushed as part of the release so they mirror the availability of `npm` packages.

The simplest use case for a typical NPM package, almost zero install downtime from ghcr and no more local tooling:

```yaml
name: "Release container"

on:
  push:
    branches:
      - main

jobs:
  release:
    environment:
      name: production
      url: https://github.com/${{ github.repository }}/releases/
tag/${{ env.RELEASE_TAG }}
    runs-on: ubuntu-latest
    steps:
      - name: "☁ checkout repository"
        uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: "🚀 release"
```

Marketplace actions should default to the major tag and are essentially more stable as we have to curate every release.

A more traditional approach, only thing really different here is a minor pull overhead and using set outputs instead of environment variables:

```yaml
name: "Release"

on:
  push:
    branches:
      - main

jobs:
  release:
    environment:
      name: production
      url: https://github.com/${{ github.repository }}/releases/
tag/${{ steps.semantic-release.outputs.release-tag }}
    name: Semantic release
    runs-on: ubuntu-latest
    steps:
      - name: "☁ checkout repository"
        uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: "🚀 release"
        id: semantic-release
        uses: open-sauced/semantic-release-conventional-config@v3
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          NPM_TOKEN: ${{ secrets.NPM_TOKEN }}

      - name: "♻ cleanup"
        run: |
```

## 📦 NPM usage

You can opt to use this package in your local tooling. Proceed as you would normally would, replacing `npm` with your package manager of choice and install the package:

```
npm install --save-dev @open-sauced/semantic-release-conventional-config
```

The shareable config can then be configured in the **semantic-release** configuration file:

```
{
  "extends": "@open-sauced/semantic-release-conventional-config"
}
```

Now all you need to do is create a release:

```
npx semantic-release
```

## 🔧 Configuration

See each plugin documentation for required installation and configuration steps.

## NPM

Set `private` to true in `package.json` if you want to disable `npm`, or, change the scope of package using `publishConfig`.

Keep one of `files` or `main` keys in your `package.json` accurate depending on whether you are building a library or an application.

If you publish, make sure to also provide a valid `NPM_TOKEN` as `.npmrc` authentication is ignored in our config!

## GitHub Actions

Unless you have an `action.yml` present in your root folder, this module is not added to the release config.

If you have an `action.yml` present, our config will attempt to adjust the container version to the newly pushed `npm` and `docker` tags.

## Docker

Unless you have a `Dockerfile` present in your root folder, this module is not added to the release config.

If you have a `Dockerfile` present, our config will attempt to push to `ghcr.io`.

## Environment variables

Using our configuration comes with some sensible defaults:

- `DOCKER_USERNAME=$GITHUB_REPOSITORY_OWNER`

- `DOCKER_PASSWORD=$GITHUB_TOKEN`
- `GIT_COMMITTER_NAME="open-sauced[bot]"`
- `GIT_COMMITTER_EMAIL="63161813+open-sauced[bot]@users.noreply.github.com"`
- `GIT_AUTHOR_NAME` - parsed from commit `$GITHUB_SHA`
- `GIT_AUTHOR_EMAIL` - parsed from commit `$GITHUB_SHA`

Feel free to change any of the above to whatever suits your purpose, our motivation is to keep `GITHUB_TOKEN` and/or `NPM_TOKEN` the only necessary requirements.

We are actively investigating ways to drop the 2 remaining variables as well!

# Workflow examples

## Node application

This example requires `"private": true,` in your `package.json` and simplifies the workflow to lightning fast deployment:

```
release:
  environment:
    name: production
    url: https://github.com/${{ github.repository }}/releases/
tag/${{ env.RELEASE_TAG }}
  name: Semantic release
  runs-on: ubuntu-latest
  steps:
    - name: "☁ checkout repository"
      uses: actions/checkout@v2
      with:
```

## Npm library

For `npm` libraries we need to set the environment URL manually and set a `NPM_TOKEN` environment variable. This also disables docker builds:

```yaml
name: "Release"

on:
  push:
    branches:
      - main

jobs:
  release:
    environment:
      name: npm
      url: https://www.npmjs.com/package/@open-sauced/
semantic-release-conventional-config/v/${{ env.RELEASE_VERSION }}
    name: Semantic release
    runs-on: ubuntu-latest
    steps:
      - name: "☁ checkout repository"
        uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: "🔧 setup node"
        uses: actions/setup-node@v2.1.5
        with:
          node-version: 16

      - name: "🔧 install npm@latest"
        run: npm i -g npm@latest
```

An up-to-date version of the example above is available at @open-sauced/semantic-release-conventional-config.

## Docker image

For docker builds it's best to build your node application in parallel with the container and re-use the artifact at a later stage:

```yaml
name: "Release"

on:
  push:
    branches:
      - main

jobs:
  docker:
    name: Build container
    runs-on: ubuntu-latest
    steps:
      - name: "☁ checkout repository"
        uses: actions/checkout@v2

      - name: "🔧 setup buildx"
        uses: docker/setup-buildx-action@v1

      - name: "🔧 cache docker layers"
        uses: actions/cache@v2
        with:
          path: /tmp/.buildx-cache
          key: ${{ runner.os }}-buildx-${{ github.sha }}
          restore-keys: |
            ${{ runner.os }}-buildx-

      - name: "🔧 docker meta"
```

An up-to-date version of the example above is available at open-sauced/open-sauced.

## Pre-releases

This workflow requires the creation of `alpha` and `beta` protected branches while templating every commit to be conventional. It does not support squashing without creating extremely complex conflict resolution:

```
name: "Release"

on:
  push:
    branches:
      - main
      - beta
      - alpha

jobs:
  release:
    environment:
      name: npm
      url: https://www.npmjs.com/package/
open-sauced-semantic-config-test/v/${{
steps.release.outputs.version }}
    name: Semantic release
    runs-on: ubuntu-latest
    steps:
      - name: "☁ checkout repository"
        uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: "🚀 release"
        id: semantic-release
```

# FAQ

## Which assets are pushed to git

The following assets are added to git using `@semantic-release/git`:

```
{
  "assets": [
    "LICENSE",
    "LICENSE.md",
    "COPYING",
    "COPYING.md",
    "CHANGELOG.md",
    "package.json",
    "package-lock.json",
    "npm-shrinkwrap.json",
    "public/diagram.svg",
    "action.yml"
  ]
}
```

## What is the commit convention

The following commit rules are enforced by `@semantic-release/commit-analyzer`:

```
{
  "preset": "conventionalcommits",
  "releaseRules": [
    { "type": "build", "release": "minor" },
```

## How to enrich the static distribution

The following assets are packed into the github release download using `@semantic-release/github`:

```
{
  "assets": [
    {
      "path": "pack/*.tgz",
      "label": "Static distribution"
    }
  ]
}
```

## How to start using pre-releases

Create the `alpha` and/or `beta` branches and protect them from being deleted or pushed to directly by non-administrators.

Switch your branching strategy to `merge` and enable conventional commits checking.

You will have to resolve merge conflicts between `alpha`, `beta` and `main` branches as described in the semantic-releases recipes.

# Contributing

We're always happy to onboard people into open source!

Check out the repository at @open-sauced/semantic-release-conventional-config ❤

# Introduction to the Chrome Extension

## The OpenSauced Chrome Extension

The OpenSauced Chrome extension seamlessly integrates GitHub with the OpenSauced platform. With this extension, you can easily view and discover open-source projects looking for contributions directly from GitHub, making collaboration and contribution easier than ever.

## Installing the Chrome Extension

To install the OpenSauced Chrome extension, navigate to the Chrome Web Store and click the "Add to Chrome" button.

# Using the Chrome Extension

## Using the OpenSauced Chrome Extension

Have you thought about using AI to refactor a code on GitHub or even using it to write the summary of your development? If you answered both of these questions as a yes, then you are in the right place. OpenSauced extension is a rich features chrome extension, That's empower developers with AI tools to:

- Using AI to Write a summary of PR.
- Get AI suggestions for code reviews
- Post your work on OpenSauced website
- Get a summary of your repo (PR and Issues stats)
- Get access to OpenSauced links
- Invite a user on OpenSauced
- View a user on OpenSauced

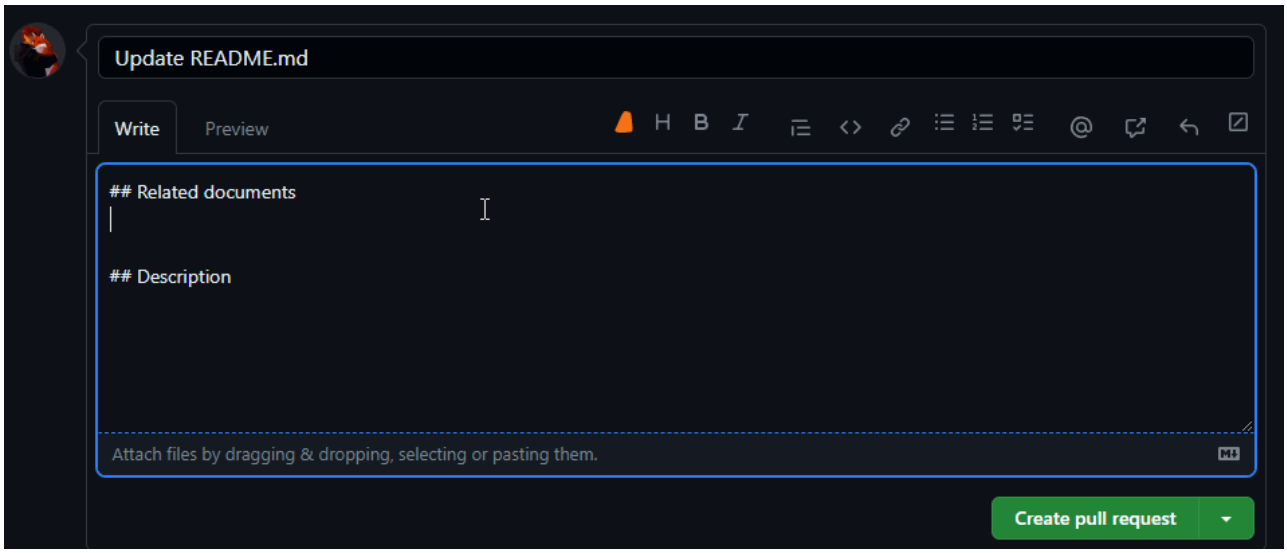
Here are some of the features that make the OpenSauced GitHub Profile Connector so useful:

### Leverage AI to generate pull request descriptions

This will help you to create a PR based on the configuration you have in the

extension, it will enable you to create the summary based on either **Commit Messages** or **The diff between the files** or **both**, you can even set the length and the tone of the message using the same settings.

*Note: This feature is not available on private repositories.*

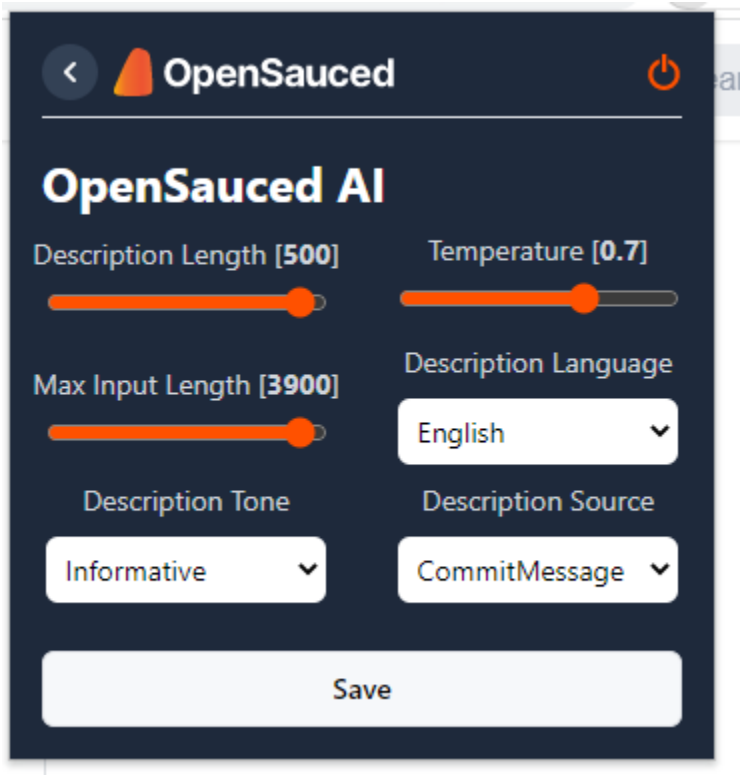



# AI Settings:

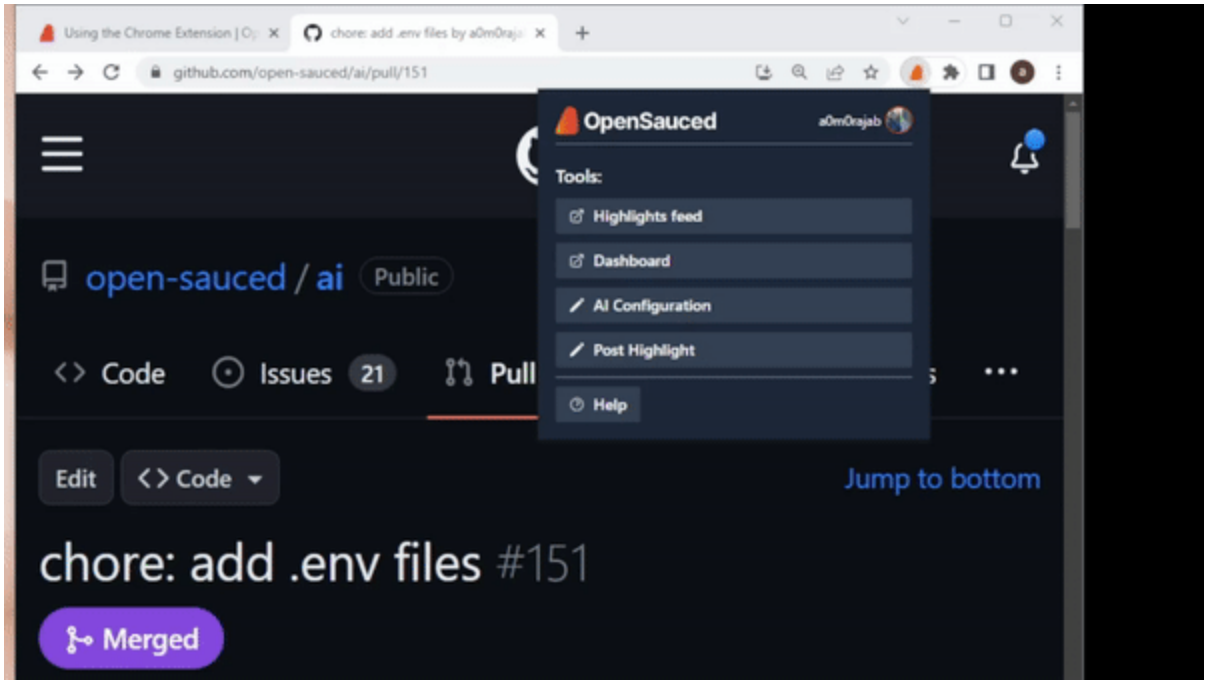
You can use this settings to set the strictness and the tone of the generated AI, here is an explanation of those settings:

- **Description** length:The output length.
- **Temperature**: Is the similarity between the input text and the output, higher temperature mean more randomness, lower temperature means more strict to the input.
- **Max Input length.**
- **Description language:** right now this feature supports: English, Spanish, French, German, Italian, Portuguese, Dutch, Russian, Chinese, Korean.
- **Description tone:** The tone of the output: Exciting, Persuasive, Informative, Humorous, Formal

- **Description source:** The source of the description that our AI will use to get the output it can be the next:**Commit Messages** or **The diff between the files** or **both**



**Changing configurations:**




**Example tones:**

Example PR, using the next options: both in the description source, 500 for output length and 3900 max input length, temperature 0.7.

**Exciting:** This PR fixes the follow call and adds a FollowUser function to the contributor-highlight-card. The code has been linted and merged from the beta branch. FollowUser function allows users to follow/unfollow contributors. Copy link and follow/unfollow options are now available in the dropdown menu.

**Persuasive:** This PR fixes the follow call and linting issues in the `ContributorHighlightCard` component. It also merges the `beta` branch into `fix-follow-check`. A new `FollowUser` function has been added to handle follow/unfollow functionality. This PR ensures a smoother user experience.

**Informative:** This PR fixes the follow call and linting issues in the

`ContributorHighlightCard` component. A new function `FollowUser` has been added to handle the follow/unfollow functionality. The `useFollowUser` hook has been moved inside this function. The `FollowUser` function is now used in place of the previous follow/unfollow code block. The code has been tested and reviewed.

**Humorous:** This PR fixes the follow call and also lints the file. We've also added a new FollowUser function to make the code more readable. Now you can follow/unfollow contributors with ease. We've also merged the beta branch into fix-follow-check.

**Formal:** This PR fixes the follow call and linting issues in the `ContributorHighlightCard` component. A new function `FollowUser` has been added to handle the follow/unfollow functionality. The `useFollowUser` hook has been moved inside this function. The `FollowUser` function is now used in place of the previous follow/unfollow code block. This PR also merges the `beta` branch into `fix-follow-check`.

# Leverage AI to generate suggestions during pull request reviews

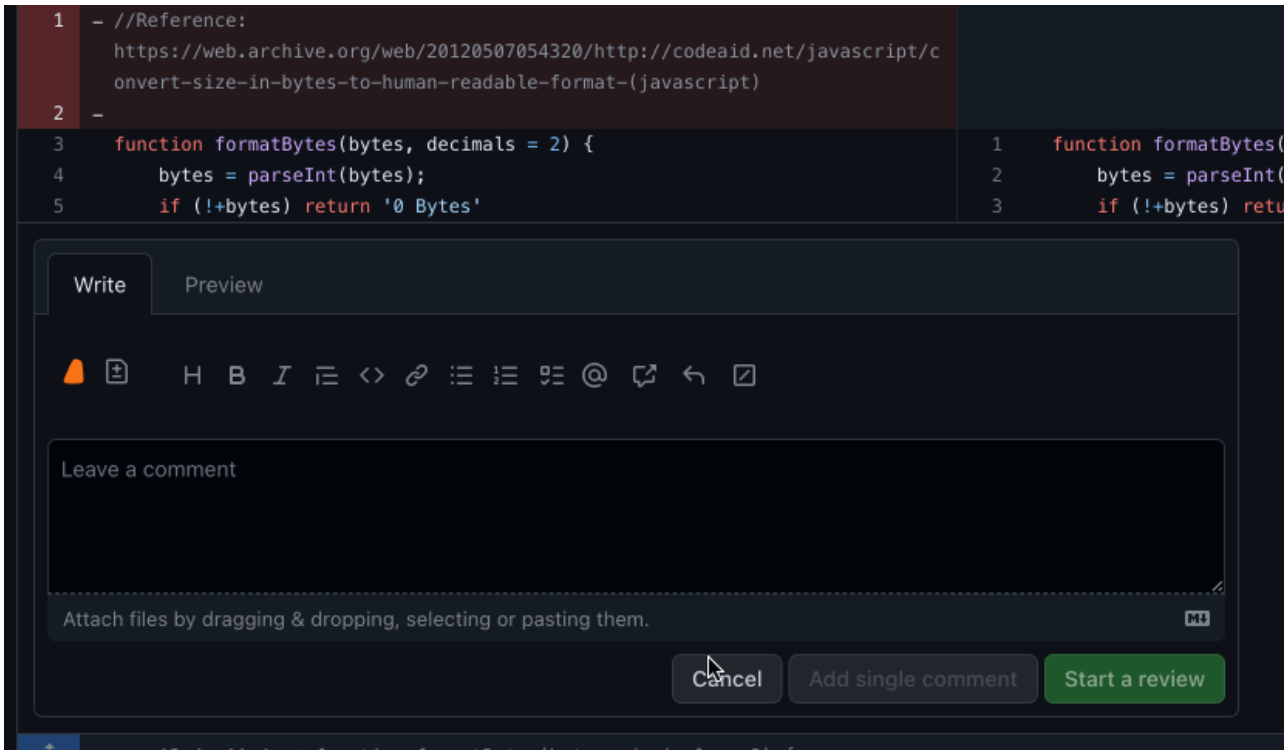
The extension enables easier PR reviews by augmenting the PR review page with AI features.

*Note: The following features are not available on private repositories.*

## AI Code Refactoring

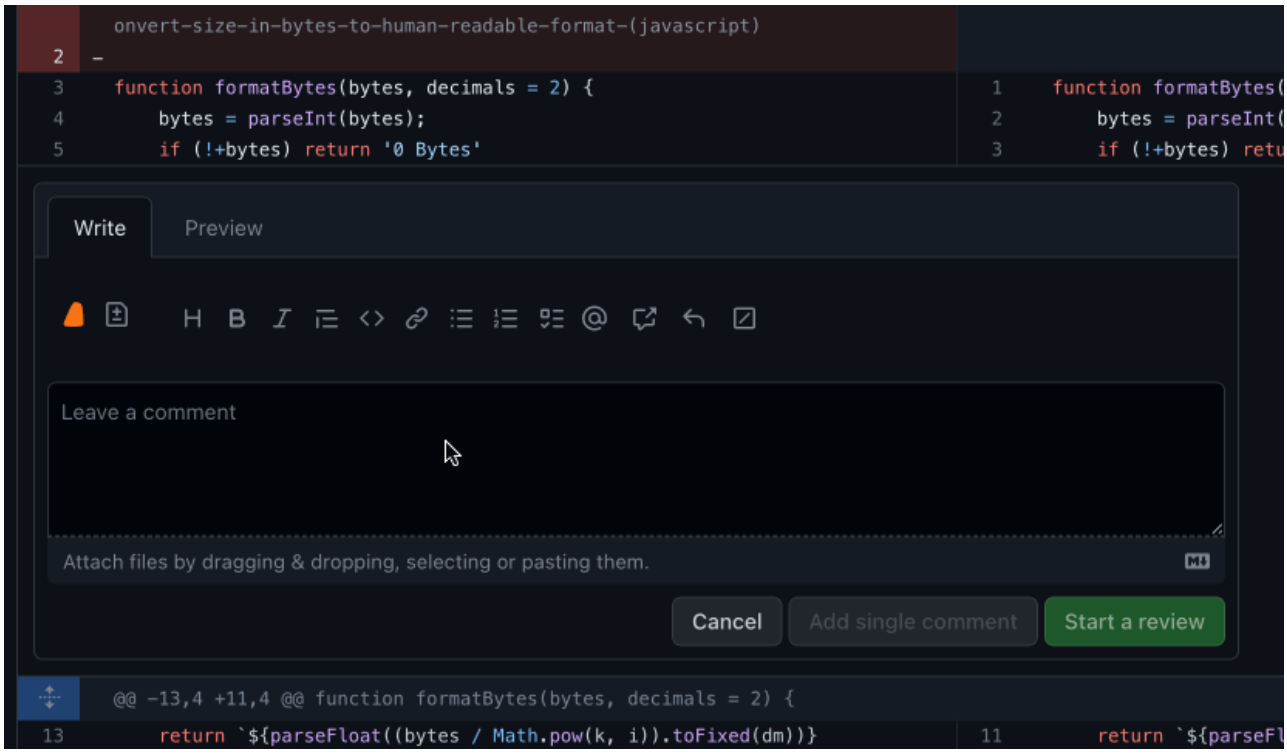
Refactor a block of code by using the blue plus button that is displayed when

hovering over a line. To select a block of code, hover over the starting line, click and drag the blue plus button until the desired line.



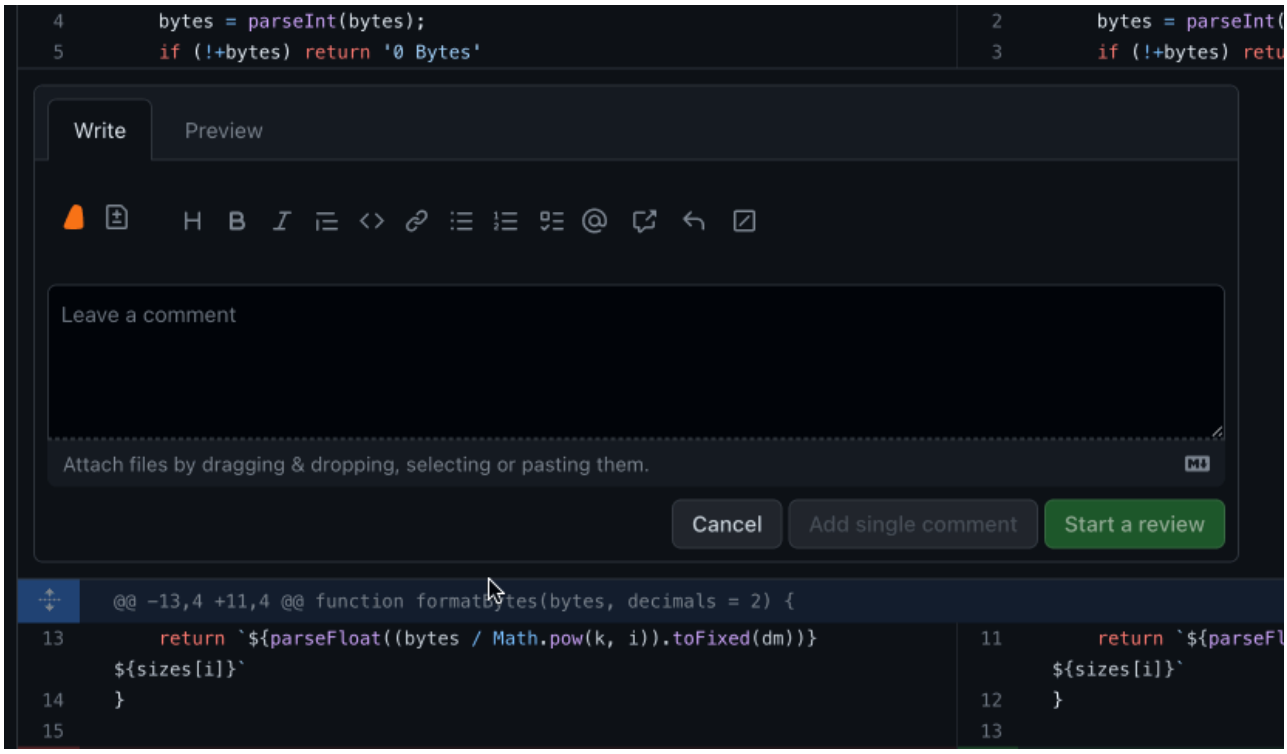

## AI Code Test Generation

Generate tests for a block of code by using the blue plus button that is displayed when hovering over a line. To select a block of code, hover over the starting line, click and drag the blue plus button until the desired line.



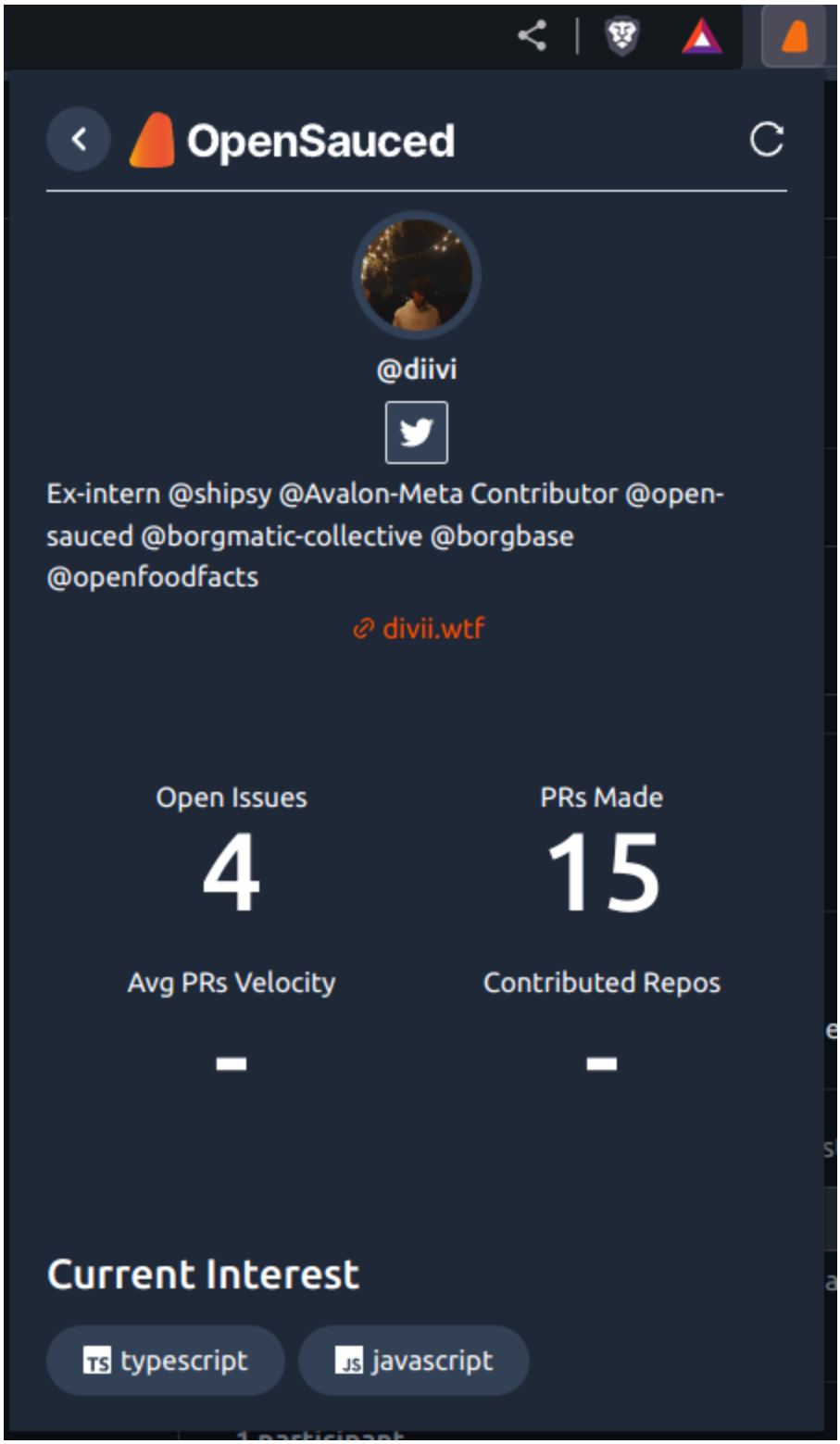

## AI Code Explanation

Generate an explanation for a block of code by using the blue plus button that is displayed when hovering over a line. To select a block of code, hover over the starting line, click and drag the blue plus button until the desired line.




## View valuable insights about your GitHub profile

View stats about open issues, PRs made, average PRs velocity, and contributed repos that are in the OpenSauced database. This page can be found by clicking the profile picture at the top left of the extension.




**Invite GitHub users to join OpenSauced with a**

## single click

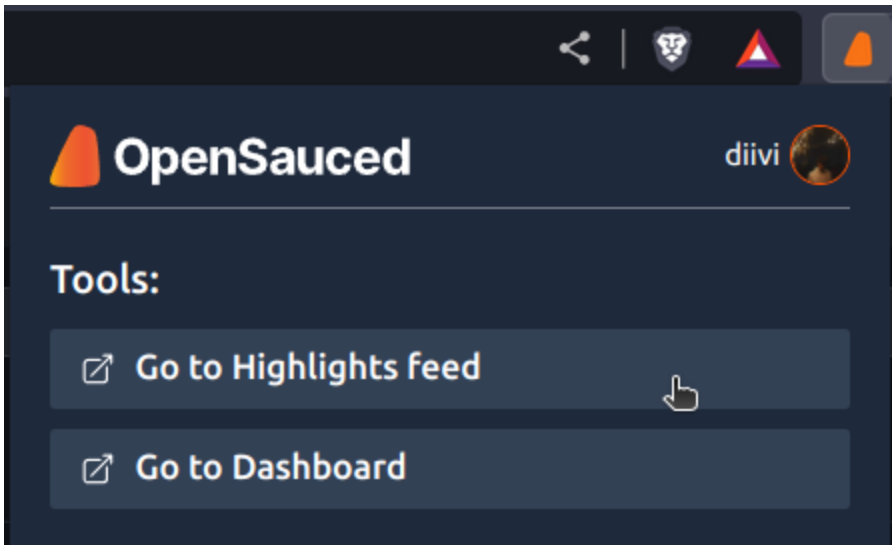
Invite other users to create an OpenSauced account to keep track of open source contributions when visiting their GitHub profile.


## View GitHub users' OpenSauced profiles and

## connect with them

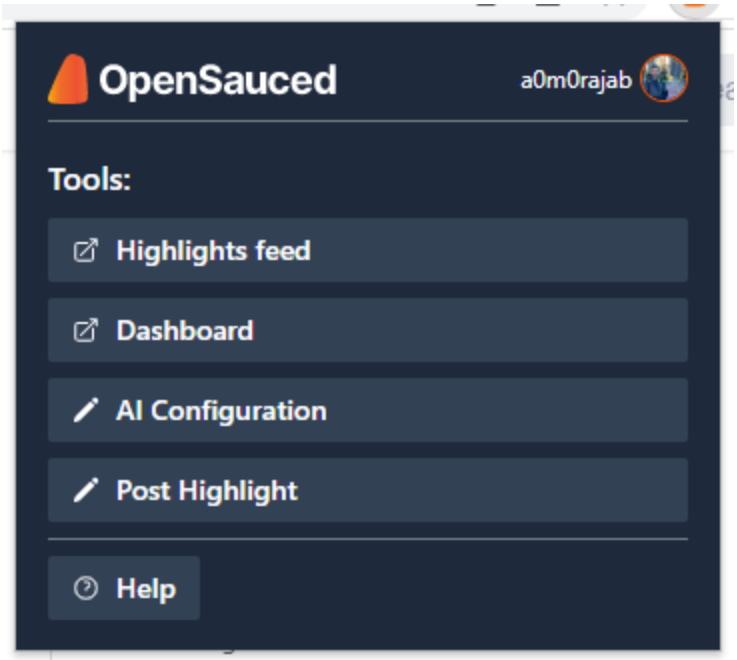
View a user’s OpenSauced profile when on their GitHub profile page.

## Quick Access to important OpenSauced links



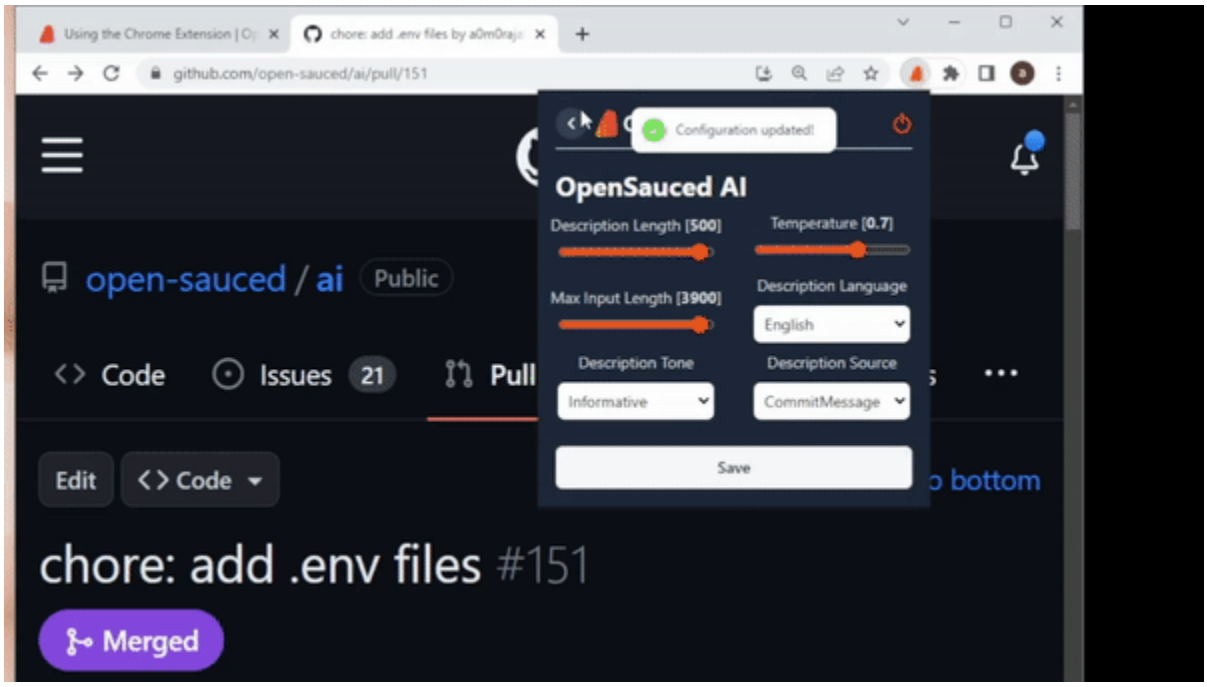

## Posting highlight

You can access the post to highlight it from the popup window in the extension. When you clicked, it will automatically, populate the PR or issue title, here you can use our AI functions to get a summary of the highlight similar to the PR summary.



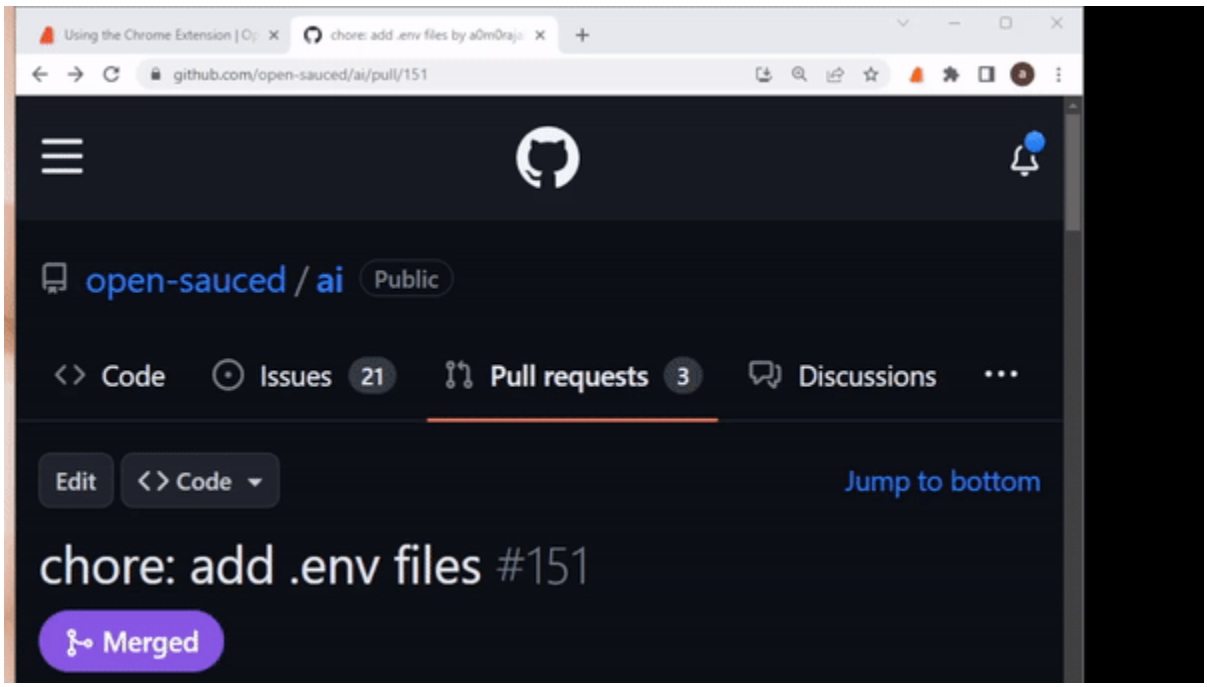

**Successful highlight:**

If you post a successful highlight it will show you a message with an option to see the highlight on the OpenSauced website.



**Errors example:**

If you presented with an error with the summarization it will be printed in the text area, this is an example of it:



# The utility functions of the project

The utility functions used by the OpenSauced Extension project can be found in src/utils. Each utility function serves a single purpose and is intended to be reusable across different parts of the project. The utilities are organized into domain-specific sub-directories within the utils folder, such as /utils/dom-utils for utilities related to manipulating the DOM.

> **NOTE: This page is not meant to be an exhaustive list of all utility functions used in the project, but to highlight a few of the ones that may otherwise be esoteric.**

## cachedFetch()

Added in PR #36, the function allows you to cache responses from API calls in local storage, reducing the number of network requests and improving performance. It takes a URL and optional options object as arguments, including an option to set the time in seconds before the cache expires and force a refresh. If there is a cached response that has not expired, it returns that instead of making another call. Otherwise, it makes a fetch request and caches the response if it meets certain criteria (status code 200 and content type JSON or text).

## createHtmlElement()

Added in PR #20, the function creates an HTML element of a specified tag name and with optional properties. The properties can include styles, which are applied to the element's inline style attribute. Other non-style properties are also supported and added to the created element using `Object.assign()`. Orginal implementation.

## checkAuthentication()

Added in PR #56, this function checks if the user is authenticated by retrieving a cookie from Chrome storage and verifying its validity. If the cookie does not exist or is invalid, it removes any saved authentication token from Chrome storage.

## domUpdateWatch()

Added in PR 64, this function is used to observe changes in the DOM and trigger a callback function when changes occur. The function uses a `MutationObserver` to track changes in the document body, and checks if the page has fully loaded before executing the callback with any specified delay time.

# Welcome to the Community

## The OpenSauced Community

Welcome to the OpenSauced community! At OpenSauced, we're striving to bring collaboration and inspiration to every open source contributor and to help build a global community of open source developers, empowering you to grow, innovate, and achieve greatness in the open.

### Community Guidelines

At OpenSauced, we strive to create a welcoming and inclusive community for everyone. We have a few guidelines to help us achieve this goal:

- Be respectful and kind to others in the community.
- Be patient with others and help them learn.
- Be open to feedback and constructive criticism.
- Adhere to the Code of Conduct.

## What's the OpenSauced Community up to?

The OpenSauced community is a group of open-source enthusiasts who are passionate about making open-source more accessible to everyone. Here are

some ways you can keep up with what we're doing:

- 🐦 Follow us on Twitter @SaucedOpen for announcements and our frequent Twitter Spaces.
- Join our Discord, and hang out with us in our weekly office hours.
- Subscribe to our YouTube channel for the latest updates and video content for OpenSauced.
- 📰 Subscribe to our newsletter for all things OpenSauced and open source.

# How can I get involved?

You can get involved in the OpenSauced community in a few ways:

- Share your Contributions! We love to see what you're working on. Highlight your contributions on OpenSauced.
- Open an issue or ask to be assigned to an existing issue on any of our OpenSauced repositories.
- Share what you're working on, ask questions, or mentor new contributors in our Discord.

# Resources on Getting Started with Open Source

- 📝 Check out our Dev blog where we provide resources for open-source contributors.
- 📖 Take our Intro to Open Source Course to help you get started with open-source.

# #100DaysOfOSS: Growing Skills and Real-World Experience

Inspired by the great work of the #100DaysOfCode challenge, we're starting #100DaysOfOSS.

With this challenge, OpenSauced hopes to help contributors enhance their skills, expand their abilities, and gain practical experience over 100 days, as well as support maintainers, onboard more contributors into open source, and expand the OSS community. With a focus on open source software (OSS), we encourage contributors of all technical backgrounds to immerse themselves in the world of collaborative development and engage with a supportive community.

## How to Participate

The beauty of this challenge is that you're not required to code. The main purpose is to grow in your understanding of open source software (OSS), contribute in ways that are meaningful to you, and further develop the skills and knowledge you're interested in pursuing. It's all about personal growth and making a positive impact on the OSS community. Because this challenge is focused on growth, you can participate in any way that helps you achieve your goals-including taking days off when you need it.

There are numerous ways to participate in the #100DaysOfOSS challenge,

including:

- write issues to identify bugs or suggest new features;
- triage existing issues to help with prioritization;
- submit pull requests to contribute code changes;
- engage in the community by sharing your insights and knowledge;
- create or participate in discussions related to OSS topics;
- write a blog post or create content that supports an OSS project;
- update or write documentation to improve clarity and usability
- create content: give a talk or presentation on OSS, participate in or even start a Twitter Space write a blog post or create a video;
- maintain a project: review pull requests, triage issues, and respond to questions;
- support contributors working on OSS projects, providing guidance and support.

To keep track of your progress, post on social media, your blog, or any other platform you prefer with what progress you made, the day of the challenge indicated by 'D'and include the hashtag #100DaysOfOSS. For example, if you're on day one, you could say, "Today, I reviewed the documentation for the OpenSauced/Insights repository. D1 #100daysOfOSS." Then, on day two, you would continue with D2, and so on.

If you're ready to join this challenge, tweet out your commitment today or share on your platform of choice!

# The Official Kickoff

We're starting 100 days from the end of Hacktoberfest, a month-long celebration of open source contributions. Starting July 23rd, we'll provide

continuous support, daily inspirational tweets, and engaging events to help you stay motivated and make progress.

Don't worry if you're unable to start on the same day as everyone else. The #100DaysOfOSS challenge is flexible, and you can join in whenever you're ready. Just jump in at any point and begin with day one of your personal challenge.

# Support

To make the most of your #100DaysOfOSS journey, here are some additional resources and events you can explore:

1. **Weekly Twitter Spaces:** Join our weekly Twitter Spaces sessions where we discuss open source topics, share insights, and connect with like-minded individuals. Follow us on Twitter to stay updated on upcoming sessions.

2. **Community Events:** Discover a wide range of events on our community docs page. Whether it's hack days, workshops, or office hours, these events provide excellent opportunities to learn, collaborate, and find new projects to contribute to.

3. **Weekly Contribution Opportunities:** If you're actively looking for open source projects to contribute to, check out the weekly post for new contribution opportunities.

4. **Weekly Office Hours:** Have questions or need help? Join us on Discord during our office hours or post in our #100DaysOfOSS channel. We're here to help you succeed!

# Where to Start?

If you're ready to start your #100DaysOfOSS journey, here are some tips to help you get started:

- **Find a Project:** Explore the OpenSauced website to find a project that interests you. You can also check out the weekly post for exciting contribution opportunities.
- **Take our Intro To Open Source Course:** If you're new to open source, we recommend taking our Intro to Open Source course to learn more about open source and how to get started.

# The Power of the #100DaysOfOSS Challenge

The #100DaysOfOSS challenge offers a supportive community where developers can find encouragement, share experiences, and overcome roadblocks together.

Our hope is that the community will provide a safe space to discuss challenges, celebrate achievements, and exchange insights, creating an environment that helps individuals stay on track and avoid giving up.

# Why join the #100DaysOfOSS Challenge:

1. **Skill Enhancement**: By working on real-world projects, you'll gain

practical experience and exposure to different projects, documentation, communities, programming languages, frameworks, and tools. You'll also learn from experienced developers, receive feedback on your code, and improve your problem-solving abilities.

2. **Collaboration and Networking**: You'll have the opportunity to work alongside other contributors, collaborate on shared goals, and build professional relationships. This experience can lead to networking opportunities, mentorship, and exposure to diverse perspectives in tech.

3. **Resume and Portfolio Boost**: Experience in open source demonstrates your ability to work in a team, follow best practices, and contribute to larger codebases. Open source contributions are tangible evidence of your skills, commitment, and ability to grow.

4. **Learning from Peers**: By examining the codebase, participating in discussions, engaging in the community, and reviewing pull requests, contributors can gain insights into different approaches to community, projects, coding styles, architecture patterns, and software development best practices, accelerating a developer's learning curve.

5. **Making a Positive Impact**: Your contributions benefit other contributors who rely on these projects, fostering a sense of fulfillment and giving back to the community.

Happy contributing and best of luck on your #100DaysOfOSS adventure!