

Introduction to Open Sauced



[Watch this on YouTube](#)

Open Sauced provides guidance for new contributors finding their next contribution. Our approach towards onboarding offers a way to track contributions through a GitHub powered dashboard.

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
```

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

Open Sauced 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 projectd 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`.

License

By contributing to the Open Sauced project, you agree that your contributions will be licensed under its [MIT 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](https://contributor-covenant.org/version/1/4), 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 opnsauced.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.

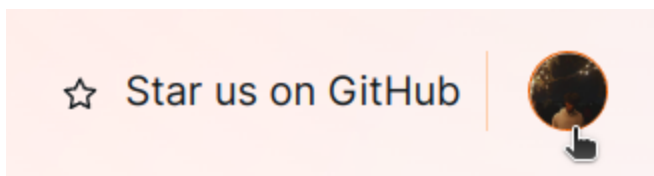
Other helpful hints:

- 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.

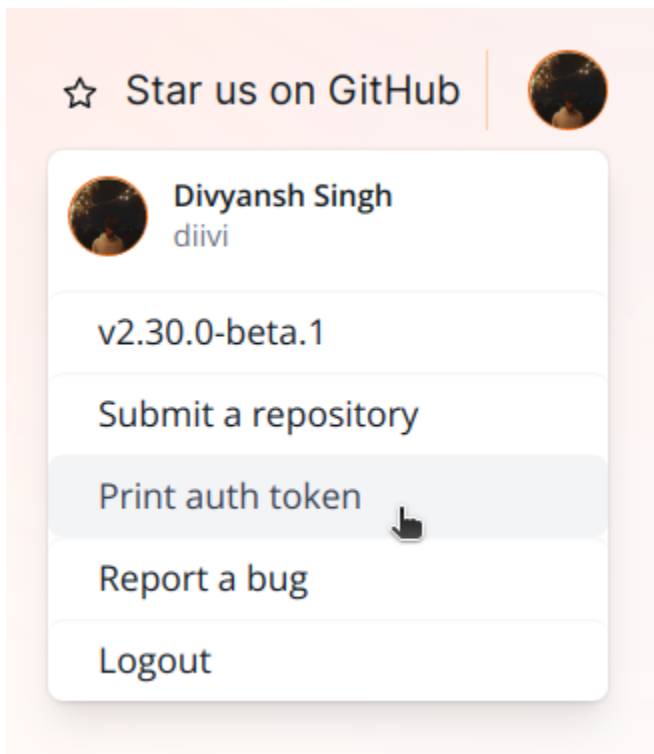
Set up Authentication

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:

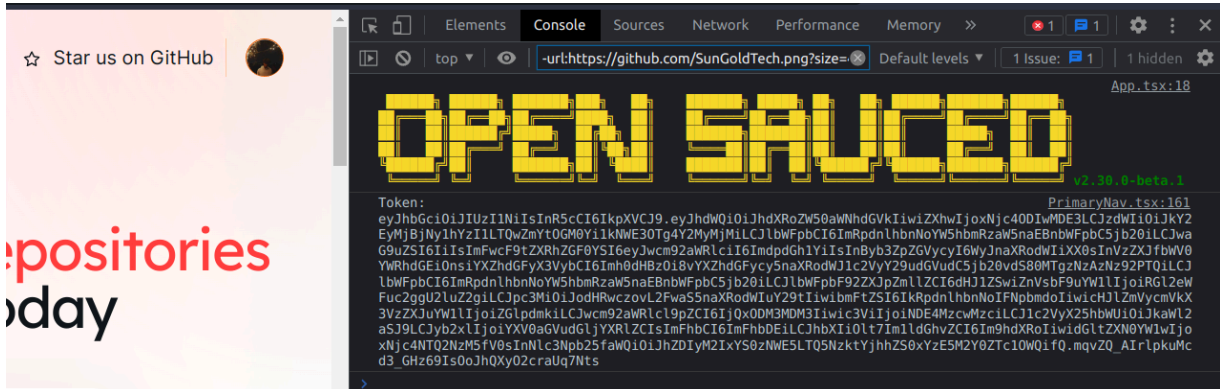
1. Click on your avatar in the top right corner of the page. This will open a dropdown menu.



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



- Copy the token that is printed to the console.



- 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/auth/session", {
  method: "GET",
  headers: {
    Authorization: `Bearer ${token}`,
  },
});
```

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* = () => (  
  
);
```

Dark mode explained

This project supports "dark mode" styling, and by default it will follow the color preference on your device. It also allows for overriding this using buttons at the top right of the screen, which will persist the preference to local storage on your device. More info about color preference web API's can be found here in the [MDN Web Docs](#).

Implementation approach

The implementation is done in 4 steps.

- The functional component in `src/containers/App.js` has a `useEffect` hook that checks the user's `localStorage` for a stored entry of `theme`. This value should be either "dark", "light", no entry at all. The hook runs only on initial page load, and if no local storage entry is found, it uses a default value of "system". It applies this value to the `ThemeContext` defined in `src/ThemeContext.js`.
- Also found in `src/containers/App.js` is a function called `systemIsDark` that can look at the user's device color preference to determine whether or not the system would prefer dark mode.
- Also found in `src/containers/App.js` is a function called `applyTheme` that checks the theme value and decides whether to include or remove the CSS classname "dark" to the `document.body` element. This determination would be made according to this table.

Value of <code>theme</code>	Result of <code>systemIsDark</code>	Apply <code>body.dark</code> ?
system	true	include
system	false	remove
dark	N/A	include
light	N/A	remove

- Found in `src/components/ThemeButtonGroup.js` is a component with three buttons, each of which can be used to change the `ThemeContext` value.

Theme context use

At this time, the `ThemeContext` only affects the `ThemeButtonGroup` component from a logic standpoint, but the `body.dark` CSS class declaration affects many places in the codebase. The broad effects of the `body.dark` CSS class declaration are found in `src/index.css`. There are more specific impacts for components that extend `styled-components` (found in `src/styles/`) and the style declarations for these refer to descendants of `body.dark`. These impacts are found in overrides of color-related properties such as `color`, `background-color`, `fill`, and `filter`.

SVG images

SVG images can have their coloring controlled by a few different means, depending on the way they're rendered. This project has some mix in its use of

SVG files for icons/images.

- In the case of its use of the library `@primer/octicons-react`, these SVG files are rendered directly into markup rather than as the `src` attribute of an `` tag, and so the `fill` CSS property is controlled by `src/index.css`.
- In the case of its use of SVG files in the repository, these are rendered as the `src` attribute of an `` tag, and therefore the `filter` CSS property is controlled by `src/index.css`.
- In the case of an SVG used as a background-image for a form element (`src/styles/Search.js`), we use an alternate SVG file for dark mode whose `fill` property has been adjusted, since this use case doesn't allow for controlling the SVG colors separately from the form element's background colors.

Loading skeletons

This project uses components from the library `react-loading-skeleton`, and these are addressed in `src/index.css` based on [this file](#) from their source code.

Best practices moving forward

For future work, components with coloring aspects should make use of the `styled-components` library and should include CSS style declarations to handle the case of `body.dark`. For example, below is a component definition used in `src/styles/TextArea.js`:

```
import styled from "styled-components";
import { margin, size, borderRadius, colors, fontSize } from
"./variables";
const Container = styled.textarea`
  margin-bottom: 12px;
  border-radius: ${borderRadius};
  border: 1px solid ${colors.lightestGrey};
  box-sizing: border-box;
  box-shadow: none;
  font-size: ${fontSize.default};
  margin-bottom: ${size.tiny};
  outline: none;
  padding: ${size.tiny};
  width: 100%;
  body.dark & {
    background-color: ${colors.darkestGrey};
    color: ${colors.lightestGrey};
  }
`;
```

References

- [Dark Mode at Stack Overflow](#)
- [Color Control of SVGs](#)
- [React Context Docs](#)

Fetching data from the GitHub graphql API

Over the past few years, GitHub has been enabling developers to build on our platform as 3rd party integrators. This enablement does not come without limitations such as rate-limiting and token access. Open Sauced originally started as a way to try out the GitHub GraphQL API with a production-ready application.

Implementation approach

Open Sauced is exclusively powered by the public data from open source repos. Not only is the data drawn from open sourced repos, the onboarding flow for Open Sauced walks the user through creating their own open sourced repo in order to track their own contributions.

OneGraph

[OneGraph](#) is the tool used to consume the GitHub GraphQL API through one consistent GraphQL interface.

Persisted queries

Persisted queries are an advanced GraphQL feature that allow clients to pre-register queries with the server. In a typical workflow, the client will send the query to the server as part of a build process and the server will send back an id for the query. When the client wants to run the query, it sends the id instead of the full query string.

Developers use persisted queries because they reduce the amount of bandwidth sent from the client to the server and they improve security by locking down the queries that can be sent to the server.

OneGraph makes this all easy to do, and you can read up more on that in [their documentation](#).

Goals repository

Each Open Sauced user leverages their own GitHub repository as a database. The repository is generated during sign up and is the companion for finding open source contributions to make. All data in this repository is a mirror of the data you see on the [opensauced.pizza](#) dashboard.

The repo [open-sauced/goals-template](#) is used as a template repo to generate the user's `open-sauced-goals` repo.

data.json The Open Sauced goal data in this list is populated using the [goals-caching.yml](#). Each opened issue in the goals repo full name is stored along with star, forks, and issues count. This is needed for rendering the user's open sauced dashboard.

stars.json The Open Sauced recommendation data is stored using the logged in user's starred repositories. This data is accessible via the GitHub API and stored publically in goals repo for easy rendering. The list is populated using the [goals-caching.yml](#). *Plans are being developed to power platform wide recommendations using this data, this is pending the reviewing of the GitHub TOCs.*

Use of API in components

The following table shows which components (`src/components/*.js`) use which API functions (`src/lib/apiGraphQL.js`, `src/lib/persistedGraphQL.js`), and what they do.

Component	API Function	Persisted/ Dynamic	Mutation
AddRepoForm	<code>api.createGoal</code> Add goal through form input	Dynamic	x
Contributions	<code>api.persistedInteractionsFetch</code> Contributions list for the user	Persisted	
CreateGoals	<code>api.fetchOwnerId</code> Need the repo owner ID first	Dynamic	
CreateGoals	<code>api.createOpenSaucedGoalsRepo</code> Create from template	Dynamic	x
DangerZone	<code>api.updateGoal</code> Remove the goal	Dynamic	x
Issues	<code>api.persistedIssuesByLabelFetch</code> Fetch Goal's issues labeled <code>good first issue</code> (First 5)	Persisted	
Issues	<code>api.persistedIssuesFetch</code>	Persisted	

Component	API Function	Persisted/ Dynamic	Mutation
	Fetch Goal's all issues (First 5)		
Issues	<code>api.persistedRepositoryIssuesByLabelFetch</code> Fetch Goal's all issues (Paginated)	Persisted	
Issues	<code>api.persistedRepositoryIssuesFetch</code> Fetch Goal's issues labeled <code>good first issue</code> (Paginated)	Persisted	
NoteForm	<code>api.updateGoal</code> Updates Notes	Dynamic	x
RecommendedRepoList	<code>api.createGoal</code> Add goal based on recommendations list	Dynamic	x
Repository	<code>api.persistedRepoDataFetch</code> Gather partial data for goal	Persisted	
Repository	<code>api.fetchGoalQuery</code> Gather partial data for goal	Dynamic	
Repository	<code>api.persistedForkFetch</code> Look at whether user has this repo forked	Persisted	
Repository	<code>api.persistedForkFetch</code> Initiate forking the repo for the user	Persisted	
RepositoryGoals	<code>api.persistedViewerStars</code> Fetch repos starred by the user	Persisted	
AdminStatsBar	<code>api.fetchRateLimit</code> Shows administrator the status of rate limiting	Dynamic	
AdminStatsBar	<code>api.persistedDeploymentFetch</code> Shows administrator deployment status	Persisted	
AdminStatsBar	<code>api.fetchRepoCount</code>	Dynamic	

Component	API Function	Persisted/ Dynamic	Mutation
	Shows administrator the user count, based on count of <code>open-sauced-goals</code> repos		

References

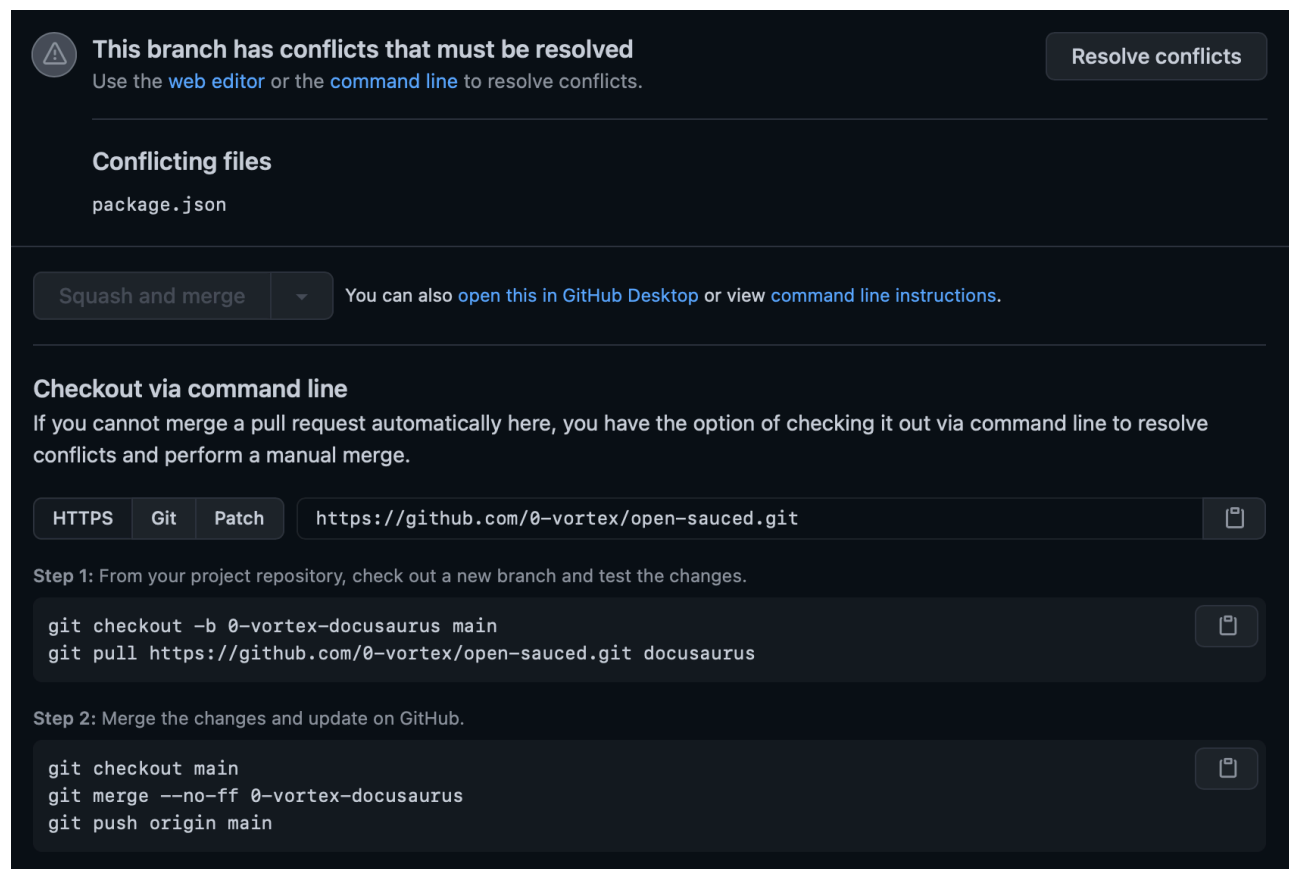
- [Using Persisted Queries](#)
- [GraphQL Enterprise Connect: "Persisted GraphQL", Brian Douglas](#)

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-shrinkwrap.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 screenshot shows the GitHub interface for resolving merge conflicts. At the top, a warning icon and text state: "This branch has conflicts that must be resolved. Use the [web editor](#) or the [command line](#) to resolve conflicts." A "Resolve conflicts" button is in the top right. Below this, the "Conflicting files" section lists "package.json". A "Squash and merge" button is on the left, and a link to "GitHub Desktop" or "command line instructions" is on the right. The "Checkout via command line" section explains that if automatic merging fails, users can check out a new branch and pull the changes. It provides two steps with terminal commands: Step 1 involves checking out a new branch and pulling the changes; Step 2 involves merging the changes and pushing them back to GitHub. Each step includes a copy icon.

This branch has conflicts that must be resolved
Use the [web editor](#) or the [command line](#) to resolve conflicts. [Resolve conflicts](#)

Conflicting files
package.json

[Squash and merge](#) [You can also open this in GitHub Desktop or view command line instructions.](#)

Checkout via command line
If you cannot merge a pull request automatically here, you have the option of checking it out via command line to resolve conflicts and perform a manual merge.

[HTTPS](#) [Git](#) [Patch](#) <https://github.com/0-vortex/open-sauced.git> [Copy](#)

Step 1: From your project repository, check out a new branch and test the changes.

```
git checkout -b 0-vortex-docusaurus main
git pull https://github.com/0-vortex/open-sauced.git docusaurus
```

Step 2: Merge the changes and update on GitHub.

```
git checkout main
git merge --no-ff 0-vortex-docusaurus
git push origin main
```

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

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 ʘ 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 ʘ (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:

```
01:28:55 in open-sauced on ʘ (git)-[docusaurus|merge]- [= $+?] via ● v16.3.0
→ git diff package.json
diff --cc package.json
index 3337138,574378c..0000000
--- a/package.json
+++ b/package.json
@@@ -85,9 -85,9 +85,15 @@@
    "build": "node scripts/build.js",
    "clean": "rimraf src/tests/__snapshots__/",
    "test": "npm run clean && node scripts/test.js --env=jsdom --updateSnapshot",
++<<<<<< HEAD
+   "storybook": "start-storybook -p 6006 --no-dll",
+   "build-storybook": "build-storybook --no-dll",
+   "lint": "eslint --ext .js,.jsx .",
++=====
+   "storybook": "start-storybook -p 6006 -s ./storybook",
+   "build-storybook": "build-storybook -s ./storybook",
+   "lint": "eslint .",
++>>>>>> origin/main
+   "lint:fix": "eslint --fix --ext .js,.jsx .",
+   "release": "standard-version --git-tag-fallback"
  },
```

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:

```
1 Merge remote-tracking branch 'origin/main' into docusaurus~
2 |
3 # By Matthew (1) and bdougie (1)~
4 # Via bdougie~
5 * origin/main:~
6   · chore(release): 0.21.22~
7   · feat: storybook dark mode (#1061)~
8 |
9 # Conflicts:~
10 #»package.json~
11 #~
12 # It looks like you may be committing a merge.~
13 # If this is not correct, please run~
14 #»git update-ref -d MERGE_HEAD~
15 # and try again.~
16 |
17 |
18 # Please enter the commit message for your changes. Lines starting~
19 # with '#' will be ignored, and an empty message aborts the commit.~
20 #~
21 # On branch docusaurus~
22 # Your branch is up to date with 'origin/docusaurus'.~
23 #~
24 # All conflicts fixed but you are still merging.~
25 #~
26 # Changes to be committed:~
27 #»modified:   .babelrc~
28 #»new file:   .storybook/index.css~
29 #»modified:   .storybook/main.js~
30 #»new file:   .storybook/preview-head.html~
31 #»new file:   .storybook/preview.js~
32 #»modified:   CHANGELOG.md~
33 #»modified:   config/polyfills.js~
34 #»modified:   npm-shrinkwrap.json~
35 #»modified:   package.json~
36 #»modified:   src/styles/Background.js~
37 #~
38 # Untracked files:~
39 #».idea/~
40 #~
41
```


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:

```
01:33:20 in open-sauced on ʦ (git)-[docusaurus|merge]- [$+?] is 📦 v0.21.22 via ● v16.3.0
→ git commit
[docusaurus 283ff8c] Merge remote-tracking branch 'origin/main' into docusaurus

01:34:30 in open-sauced on ʦ docusaurus [!$?] is 📦 v0.21.22 via ● v16.3.0 took 1m 5s
→ git status
On branch docusaurus
Your branch is ahead of 'origin/docusaurus' by 3 commits.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .idea/

nothing added to commit but untracked files present (use "git add" to track)

01:34:42 in open-sauced on ʦ docusaurus [!$?] is 📦 v0.21.22 via ● v16.3.0
→ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.34 KiB | 98.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:0-vortex/open-sauced.git
 f4c7665..283ff8c  docusaurus → docusaurus

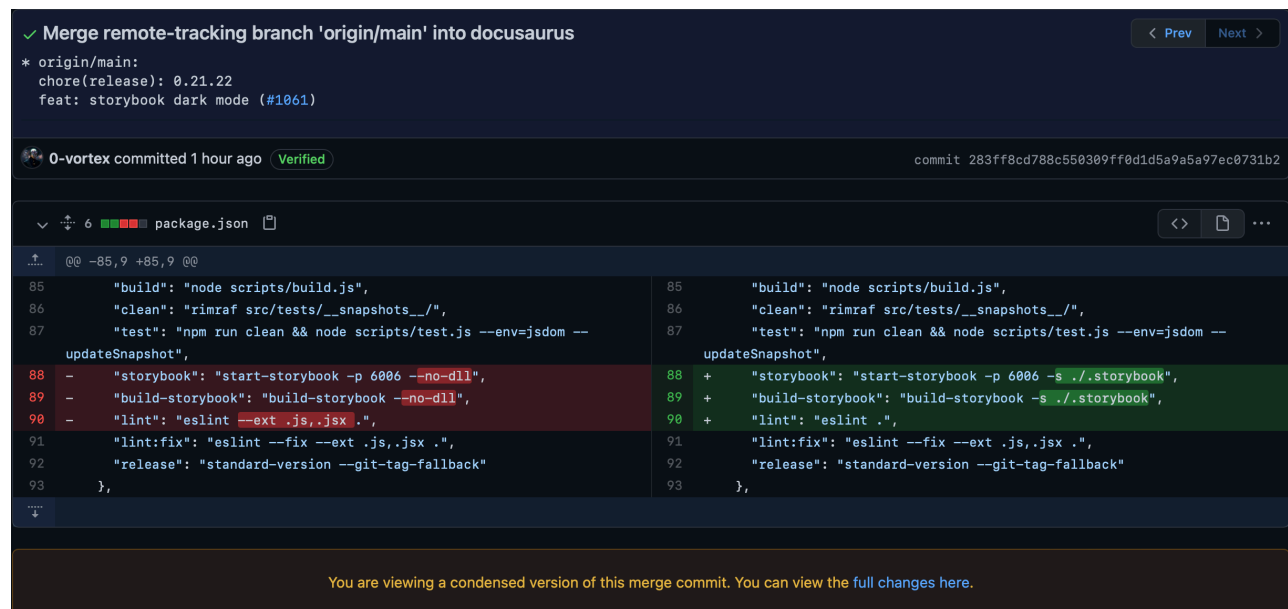
01:35:25 in open-sauced on ʦ docusaurus [$?] is 📦 v0.21.22 via ● v16.3.0 took 4s
→
```

Review your pull request

The result of the above commands can be viewed at

[283ff8cd788c550309ff0d1d5a9a5a97ec0731b2](https://github.com/docusaurus/docusaurus/commit/283ff8cd788c550309ff0d1d5a9a5a97ec0731b2)

GitHub will conveniently display only you merge conflict changes:



The screenshot shows a GitHub pull request interface. At the top, a green checkmark indicates a successful merge of the remote-tracking branch 'origin/main' into 'docusaurus'. Below this, the commit details for '0-vortex' are shown, including the commit hash '283ff8cd788c550309ff0d1d5a9a5a97ec0731b2'. The main part of the image is a diff view for the 'package.json' file. It shows a side-by-side comparison of the file's content. The left side represents the original code, and the right side represents the changes. The changes are highlighted with a green background. The diff shows several modifications to the 'scripts' section of the 'package.json' file, including updates to the 'build', 'clean', 'test', 'storybook', 'build-storybook', 'lint', 'lint:fix', and 'release' scripts. A message at the bottom of the diff view states: 'You are viewing a condensed version of this merge commit. You can view the full changes here.'

```
✓ Merge remote-tracking branch 'origin/main' into docusaurus
* origin/main:
 chore(release): 0.21.22
 feat: storybook dark mode (#1061)

0-vortex committed 1 hour ago Verified commit 283ff8cd788c550309ff0d1d5a9a5a97ec0731b2

package.json
@@ -85,9 +85,9 @@
 85     "build": "node scripts/build.js",
 86     "clean": "rimraf src/tests/__snapshots__/",
 87     "test": "npm run clean && node scripts/test.js --env=jsdom --
      updateSnapshot",
88 -   "storybook": "start-storybook -p 6006 --no-dll",
89 -   "build-storybook": "build-storybook --no-dll",
90 -   "lint": "eslint --ext .js,.jsx .",
91   "lint:fix": "eslint --fix --ext .js,.jsx .",
92   "release": "standard-version --git-tag-fallback"
93 },
+   "storybook": "start-storybook -p 6006 -s ../.storybook",
+   "build-storybook": "build-storybook -s ../.storybook",
+   "lint": "eslint .",
91   "lint:fix": "eslint --fix --ext .js,.jsx .",
92   "release": "standard-version --git-tag-fallback"
93 },
```

You are viewing a condensed version of this merge commit. You can view the full changes here.

And it's ready to merge:

✓ All checks have passed

7 successful, 3 skipped, and 3 neutral checks

Hide all checks

✓	CodeQL / Analyze (javascript) (pull_request)	Successful in 1m	Details
✓	Node CI/CD / BUILD (pull_request)	Successful in 1m	Details
✓	PR Audit / semantics (pull_request_target)	Successful in 7s	Details
⊘	Node CI/CD / Documentation (pull_request)	Skipped	Details
⊘	PR Audit / audit (pull_request_target)	Skipped	Details
⊘	PR Audit / welcome (pull_request_target)	Skipped	Details

✓ This branch has no conflicts with the base branch

Merging can be performed automatically.

Squash and merge

You can also open this in GitHub Desktop or view command line instructions.

Dependency updates

When dealing with dependency and lock file updates there are multiple use cases to consider, however as a baseline, the open sauced 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 opnsauced.pizza
2. Join [discord](https://discord.com).
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](https://open-sauced.com/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!

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *



open-sourced ▾

Repository name *

npx-check-engines



Great repository names are short and memorable. Need inspiration? How about **verbose-chainsaw**?

Description (optional)

Never break your dependency tree with npm-install-checks running on `npx`



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You may not create private repositories by organization policy.

Initialize this repository with:

Skip this step if you're importing an existing repository.



Add a README file

This is where you can write a long description for your project. [Learn more](#).



Add .gitignore


Choose which files not to track from a list of templates. [Learn more](#).



Choose a license

A license tells others what they can and can't do with your code. [Learn more](#).

License: MIT License ▾

This will set  `main` as the default branch. Contact the organization admin to change the default name.

Syncing settings with

opensauced.pizza

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

Merge button

When merging pull requests, you can allow any combination of merge commits, squashing, or rebasing. At least one option must be enabled. If you have linear history requirement enabled on any protected branch, you must enable squashing or rebasing.

☐ **Allow merge commits**

Add all commits from the head branch to the base branch with a merge commit.

☒ **Allow squash merging**

Combine all commits from the head branch into a single commit in the base branch.

☒ **Allow rebase merging**

Add all commits from the head branch onto the base branch individually.

You can allow setting pull requests to merge automatically once all required reviews and status checks have passed.

☐ **Allow auto-merge**

Waits for merge requirements to be met and then merges automatically. [Learn more](#)

After pull requests are merged, you can have head branches deleted automatically.

☒ **Automatically delete head branches**

Deleted branches will still be able to be restored.

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.

Branch name pattern

main

Protect matching branches

☒ **Require pull request reviews before merging**

When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.

Required approving reviews: 1 ▾

☒ **Dismiss stale pull request approvals when new commits are pushed**

New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

☒ **Require review from Code Owners**

Require an approved review in pull requests including files with a designated code owner.

☐ **Restrict who can dismiss pull request reviews**

Specify people or teams allowed to dismiss pull request reviews.

☒ **Require status checks to pass before merging**

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☒ **Require branches to be up to date before merging**

This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Q Search for status checks in the last week for this repository

Status checks that are required.

☒ **Require conversation resolution before merging**

When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. [Learn more](#).

☒ **Require signed commits**

Commits pushed to matching branches must have verified signatures.

☒ **Require linear history**

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.

Environments / Configure npm

Environment protection rules

Can be used to configure manual approvals and timeouts.

☐ **Required reviewers**

Specify people or teams that may approve workflow runs when they access this environment.

☐ **Wait timer**

Set an amount of time to wait before allowing deployments to proceed.

Save protection rules

Deployment branches

Can be used to limit what branches can deploy to this environment using branch name patterns.

Selected branches ▾

1 branch allowed

main

Currently applies to 1 branch

Edit

Remove

+ Add deployment branch rule

Environment secrets

Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment.

🔒 NPM_TOKEN

Updated 4 hours ago

Update

Remove

+ Add Secret

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](https://github.com/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 corresponding 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`
- `@eclclass/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:

```
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"
        id: semantic-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:

```
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"
```



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:

```
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:

```
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-
```

An up-to-date version of the example above is available at [open-sauced/open-sauced](https://github.com/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"
```

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](#) ♥

