Codemods

Codemods are transformations that run on your codebase programmatically. This allows a large number of changes to be programmatically applied without having to manually go through every file.

Next.js provides Codemod transformations to help upgrade your Next.js codebase when an API is updated or deprecated.

Usage

In your terminal, navigate (`cd`) into your project's folder, then run:

Terminal

```
npx @next/codemod <transform> <path>
```

Replacing `<transform>` and `<path>` with appropriate values.

- `transform` - name of transform
- `path` - files or directory to transform
- `--dry` Do a dry-run, no code will be edited
- `--print` Prints the changed output for comparison

Next.js Codemods

13.2

Use Built-in Font

`built-in-next-font`

Terminal

```
npx @next/codemod@latest built-in-next-font
```

This codemod uninstalls the `@next/font` package and transforms `@next/font` imports into the built-in `next/font`.

For example:

```
import { Inter } from '@next/font/google';
```

Transforms into:

```
import { Inter } from 'next/font/google';
```

13.0

Rename Next Image Imports

`next-image-to-legacy-image`

Terminal

```
npx @next/codemod@latest next-image-to-legacy-image ./pages
```

Safely renames `next/image` imports in existing Next.js 10, 11, or 12 applications to `next/legacy/image` in Next.js 13. Also renames `next/future/image` to `next/image`.

For example:

pages/index.js

```
import Image1 from 'next/image';
import Image2 from 'next/future/image';

export default function Home() {
  return (
    <div>
      <Image1 src="/test.jpg" width="200" height="300" />
      <Image2 src="/test.png" width="500" height="400" />
    </div>
  );
}
```

Transforms into:

pages/index.js

```jsx
// 'next/image' becomes 'next/legacy/image'
import Image1 from 'next/legacy/image';
// 'next/future/image' becomes 'next/image'
import Image2 from 'next/image';

export default function Home() {
  return (
    <div>
      <Image1 src="/test.jpg" width="200" height="300" />
      <Image2 src="/test.png" width="500" height="400" />
    </div>
  );
}
```

Migrate to the New Image Component

`next-image-experimental`

Terminal

```
npx @next/codemod@latest next-image-experimental ./pages
```

Dangerously migrates from `next/legacy/image` to the new `next/image` by adding inline styles and removing unused props.

- Removes `layout` prop and adds `style`.
- Removes `objectFit` prop and adds `style`.
- Removes `objectPosition` prop and adds `style`.
- Removes `lazyBoundary` prop.
- Removes `lazyRoot` prop.

Remove `<a>` Tags From Link Components

`new-link`

Terminal

```
npx @next/codemod@latest new-link ./pages
```

Remove `<a>` tags inside Link Components, or add a `legacyBehavior` prop to Links that cannot be auto-fixed.

For example:

```
<Link href="/about">
  <a>About</a>
</Link>
// transforms into
<Link href="/about">
  About
</Link>

<Link href="/about">
  <a onClick={() => console.log('clicked')}>About</a>
</Link>
// transforms into
<Link href="/about" onClick={() => console.log('clicked')}>
  About
</Link>
```

In cases where auto-fixing can't be applied, the `legacyBehavior` prop is added. This allows your app to keep functioning using the old behavior for that particular link.

```
const Component = () => <a>About</a>

<Link href="/about">
  <Component />
</Link>
// becomes
<Link href="/about" legacyBehavior>
  <Component />
</Link>
```

11

Migrate from CRA

`cra-to-next`

Terminal

```
npx @next/codemod cra-to-next
```

Migrates a Create React App project to Next.js; creating a Pages Router and necessary config to match behavior. Client-side only rendering is leveraged initially to prevent breaking compatibility due to `window` usage during SSR and can be enabled seamlessly to allow the gradual adoption of Next.js specific features.

Please share any feedback related to this transform in this discussion ↗.

10

Add React imports

`add-missing-react-import`

Terminal

```
npx @next/codemod add-missing-react-import
```

Transforms files that do not import `React` to include the import in order for the new React JSX transform ↗ to work.

For example:

my-component.js

```js
export default class Home extends React.Component {
  render() {
    return <div>Hello World</div>;
  }
}
```

Transforms into:

my-component.js

```js
import React from 'react';
export default class Home extends React.Component {
  render() {
    return <div>Hello World</div>;
  }
}
```

9

Transform Anonymous Components into Named Components

`name-default-component`

Terminal

```
npx @next/codemod name-default-component
```

Versions 9 and above.

Transforms anonymous components into named components to make sure they work with Fast Refresh ↗.

For example:

my-component.js

```js
export default function () {
  return <div>Hello World</div>;
}
```

Transforms into:

my-component.js

```js
export default function MyComponent() {
  return <div>Hello World</div>;
}
```

The component will have a camel-cased name based on the name of the file, and it also works with arrow functions.

8

Transform AMP HOC into page config

`withamp-to-config`

Terminal

```
npx @next/codemod withamp-to-config
```

Transforms the `withAmp` HOC into Next.js 9 page configuration.

For example:

```js
// Before
```

```
import { withAmp } from 'next/amp';

function Home() {
  return <h1>My AMP Page</h1>;
}

export default withAmp(Home);
```

```
// After
export default function Home() {
  return <h1>My AMP Page</h1>;
}

export const config = {
  amp: true,
};
```

6

Use `withRouter`

`url-to-withrouter`

Terminal

```
npx @next/codemod url-to-withrouter
```

Transforms the deprecated automatically injected `url` property on top level pages to using `withRouter` and the `router` property it injects. Read more here: https://nextjs.org/docs/messages/url-deprecated

For example:

From

```
import React from 'react';
export default class extends React.Component {
  render() {
    const { pathname } = this.props.url;
    return <div>Current pathname: {pathname}</div>;
  }
}
```

To

```
import React from 'react';
import { withRouter } from 'next/router';
```

```
export default withRouter(
  class extends React.Component {
    render() {
      const { pathname } = this.props.router;
      return <div>Current pathname: {pathname}</div>;
    }
  },
);
```

This is one case. All the cases that are transformed (and tested) can be found in the `__testfixtures__` directory ↗.