Looking for the v1 docs? Find them here.

> *This document is a work in progress. Have you run into something that's not covered here? Add your changes on GitHub!*

Introduction

This is a reference for upgrading your site from Gatsby v1 to Gatsby v2. While there's a lot covered here, you won't need to do everything for your site. We'll do our best to keep things easy to follow, and as sequential as possible so you can get rocking on v2!

> *If you want to start fresh, check out the starting a new project section*

Why you should migrate

This documentation page covers the *how* of migrating from v1 to v2. Various blog posts cover the *why*:

- v2 Overview by Kyle Mathews
- Improving accessibility by Amberley Romo
- Keeping Gatsby sites blazing fast by Dustin Schau

What we'll cover

Updating Your Dependencies

First, you need to update your dependencies and install any needed peer dependencies.

Update Gatsby version

You need to update your `package.json` to use the latest version of Gatsby.

```
"dependencies": {
  "gatsby": "^2.0.0",
}
```

Or run

```
npm install gatsby@latest
```

Update Gatsby related packages

Update your `package.json` to use the latest versions of Gatsby related packages. You should upgrade any package name that starts with `gatsby-`. Note, this only applies to plugins managed in the gatsbyjs/gatsby repo. If you're using community plugins, they might not be upgraded yet. Check their repo for the status. Many plugins won't need upgrading so they very well might keep working. You can run

```
npm outdated
```

And compare "Wanted" and "Latest" versions and update `package.json` file manually or run

```
npm install gatsby-plugin-google-analytics@latest gatsby-plugin-netlify@latest
gatsby-plugin-sass@latest
```

NOTE: The above command is only an example - adjust packages to ones you are using.

Install React

In v1, the `react` and `react-dom` packages were a part of the `gatsby` package. They are now `peerDependencies` in v2, so you need to install them into your project.

```
npm install react react-dom
```

Install plugins' peer dependencies

Some plugins had dependencies that were also made `peerDependencies`. For example, if you use `gatsby-plugin-typography`, you now need to install:

```
npm install typography react-typography
```

You should search for the plugins that you use in the plugin library. Then, check their installation instructions for extra packages that may need installation.

Handling Breaking Changes

Remove or refactor layout components

Gatsby's layout components (`src/layouts/index.js`) are gone now. The "top-level component" is now the page itself. If the layout of your site looks broken, this is likely the reason why.

There are some implications to this change:

- Rendering different layouts for different pages is different. Use the standard React inheritance model. Gatsby no longer maintains, or needs to maintain, separate behavior for handling layouts.
- Because the "top-level component" changes between each page, React will rerender all children. This means that shared components previously in a Gatsby v1 layout-- like navigations-- will unmount and remount. This will break CSS transitions or React state within those shared components. If your use case requires a layout component to not unmount use `gatsby-plugin-layout`.
- To learn more about the decision behind this removal, read the RFC for removing the special layout component.

We recommend the following migration path:

1. Convert the layout's children from a render prop to a normal prop (required)

In v1, the `children` prop passed to the layout was a function (render prop) and needed to be executed. In v2, this is no longer the case.

```
import React from "react"
```

```diff
export default function Layout({ children }) {
  return (
    <div>
-     {children()}
+     {children}
    </div>
  );
}
```

2. Move `layouts/index.js` to `src/components/layout.js` (optional, but recommended)

```
git mv src/layouts/index.js src/components/layout.js
```

3. Import and wrap pages with the layout component

Adhering to the normal React composition model, import your layout component, and use it to wrap the content of the page.

```
import React from "react"
import Layout from "../components/layout"

export default function Home() {
  return (
    <Layout>
      <div>Hello World</div>
    </Layout>
  )
}
```

Repeat for every page and template that needs this layout.

4. Pass `history` , `location` , and `match` props to layout

In v1, the layout component had access to `history` , `location` , and `match` props. In v2, only pages have access to these props. If you need these props in the layout component, pass them through from the page.

```
import React from "react"

export default function Layout({ children, location }) {
  return (
    <div>
      <p>Path is {location.pathname}</p>
      {children}
    </div>
  )
}
```

```
import React from "react"
import Layout from "../components/layout"
```

```
export default function Home(props) {
  return (
    <Layout location={props.location}>
      <div>Hello World</div>
    </Layout>
  )
}
```

5. Change the query to use `StaticQuery`

If you were using the `data` prop in your Gatsby v1 layout, you now need to use Gatsby v2's StaticQuery feature. This is because a layout is now a normal component.

Replacing a layout's query with `StaticQuery`:

```
import React, { Fragment } from "react"
import { Helmet } from "react-helmet"
+ import { StaticQuery, graphql } from "gatsby"

- export default function Layout({ children, data }) {
-   return (
-     <>
-       <Helmet titleTemplate={`%s | ${data.site.siteMetadata.title}`} defaultTitle=
{data.site.siteMetadata.title} />
-       <div>
-         {children()}
-       </div>
-     </>
-   );
- }
-
- export const query = graphql`
-   query LayoutQuery {
-     site {
-       siteMetadata {
-         title
-       }
-     }
-   }
- `
+ export default function Layout({ children }) {
+   return (
+     <StaticQuery
+       query={graphql`
+         query LayoutQuery {
+           site {
+             siteMetadata {
+               title
+             }
+           }
+         }
+       `}
+        render={data => (
+          <>
+            <Helmet titleTemplate={`%s | ${data.site.siteMetadata.title}`}
defaultTitle={data.site.siteMetadata.title} />
+            <div>
+              {children}
+            </div>
+          </>
+        )}
+      />
+    );
+ }
```

Change `navigateTo` to `navigate`

The `navigateTo` method in `gatsby-link` was renamed to `navigate` to mirror the API used by @reach/router.

In addition to the name change, `gatsby-link` is now exported from the `gatsby` package and can't be installed directly.

```
import React from "react"
- import { navigateTo } from "gatsby-link"
+ import { navigate } from "gatsby"

// Don't use navigate with an onClick btw :-)
// Generally use the `<Link>` component.
export default function Page(props) {
  return (
-     <div onClick={() => navigateTo(`/`)}>Click to go to home</div>
+     <div onClick={() => navigate(`/`)}>Click to go to home</div>
  )
}
```

Convert to either pure CommonJS or pure ES6

Gatsby v2 uses webpack 4 which is stricter about modules with mixed module systems.

All ES6 is **GOOD**:

```
// GOOD: ES modules syntax works
import foo from "foo"
export default foo
```

All CommonJS is **GOOD**:

```
// GOOD: CommonJS syntax works
const foo = require("foo")
module.exports = foo
```

Mixing `requires` and `export` is **BAD**:

```
// BAD: Mixed ES and CommonJS module syntax will cause failures
const foo = require("foo")
export default foo
```

Mixing `import` and `module.exports` is **BAD**:

```
// BAD: Mixed ES and CommonJS module syntax will cause failures
import foo from "foo"
module.exports = foo
```

Move Babel Configuration

The latest version of Gatsby uses Babel 7. Babel 7 introduced a new behavior for configuration lookup / resolution. In the case where a `.babelrc` file might be used at the root of the project (like for configuring Jest), moving that Babel configuration into *jest.config.json* will avoid any conflicts.

This GitHub comment documents the steps needed to do that.

More information on Gatsby and Babel configuration available here.

Restore v1 PostCSS Plugin Setup

Gatsby v2 removed `postcss-cssnext` and `postcss-import` from the default PostCSS setup.

Use `gatsby-plugin-postcss` to have the same configuration that you had in v1.

1. Install the dependencies

`npm install gatsby-plugin-postcss postcss-import postcss-cssnext postcss-browser-reporter postcss-reporter`

NOTE: `postcss-cssnext` is deprecated and it is better to use `postcss-preset-env`.

2. Include `gatsby-plugin-postcss` in your `gatsby-config.js` file

```
plugins: [`gatsby-plugin-postcss`],
```

3. Include PostCSS plugins in your `postcss.config.js` file

```
const postcssImport = require(`postcss-import`)
const postcssCssNext = require(`postcss-cssnext`)
const postcssBrowserReporter = require(`postcss-browser-reporter`)
const postcssReporter = require(`postcss-reporter`)

module.exports = () => ({
  plugins: [
    postcssImport(),
    postcssCssNext(),
    postcssBrowserReporter(),
    postcssReporter(),
  ],
})
```

Migrate from React Router to @reach/router

We switched our router from React Router v4 to @reach/router as @reach/router is smaller and has 1st class support for accessibility.

The founder of React Router, Ryan Florence is also the founder of @reach/router.

He says @reach/router restores things he misses from React Router v3. @reach/router also retains the best parts of React Router v4 *and* adds full accessibility support.

For *most* sites, this change won't cause any breaking changes as the two routers are very similar.

Two common ways this change *might* break your site is:

- You use the object form of the `to` prop in the `<Link>` component
- You have client-side routes

Read more about the features of our new router at https://reach.tech/router

NOTE: One prominent feature of @reach/router, relative routes, isn't working currently in Gatsby. We're working with Ryan Florence on hopefully supporting it soon.

Read on for instructions on migrating your site to @reach/router.

Only string `to` allowed

React Router allowed you to pass objects to the `to` prop e.g.

```
<Link
  to={{ pathname: `/about/`, search: `fun=true&pizza=false`, hash: `people` }}
>
  Our people
</Link>
```

React Router would then concatenate the object values together into the full pathname e.g. `/about/?fun=true&pizza=false#people`.

Now you'll need to concatenate together the full pathname yourself.

```diff
- <Link to={{ pathname: `/about/`, search: `fun=true&pizza=false`, hash:
`people`}}>Our people</Link>
+ <Link to={`/about/?fun=true&pizza=false#people`}>Our people</Link>
```

Pass state to the `state` prop

Before with React Router to pass state to a link, you would pass it as part of a `to` object prop.

Now, to add state to a link, pass it via a `state` prop.

```
const NewsFeed = () => (
  <div>
    <Link to="photos/123" state={{ fromFeed: true }} />
  </div>
)

const Photo = ({ location, photoId }) => {
  if (location.state.fromFeed) {
    return <FromFeedPhoto id={photoId} />
  } else {
    return <Photo id={photoId} />
  }
}
```

A `history` prop is no longer passed to page components

React Router would pass a `history` prop to components that you could use to navigate.

If you need to do programmatic navigation, import @reach/router's `navigate` function instead.

```
import { navigate } from "@reach/router"
```

The following props are no longer available on `<Link>`

- `exact`
- `strict`
- `location`

`exact` and `strict` are no longer necessary as @reach/router does matching this way by default.

You could pass `location` before to manually compute whether the link is active or not. For advanced link stylings, use `getProps` now.

Use `getProps` for advanced link styling

Gatsby's `<Link>` component supports out-of-the-box `activeClassName` and `activeStyle`.

If you have more advanced styling needs, use the `getProps` prop.

Change client paths to use a splat

When creating a client route in `gatsby-node.js`, use a `*` to select all child routes instead of `:path`.

```
exports.onCreatePage = async ({ page, actions }) => {
  const { createPage } = actions

  // page.matchPath is a special key that's used for matching pages
  // only on the client.
  if (page.path.match(/^\/app/)) {
-    page.matchPath = "/app/:path"
+    page.matchPath = "/app/*"
```

```
    // Update the page.
    createPage(page)
  }
}
```

Migrating React Router client routes to @reach/router

- Use `<Location>` instead of `withRouter`
- `import { navigate } from @reach/router` for programmatic navigation instead of the history object
- There's no `Route` component anymore. You can add a `<Router>` component (a site can have as many routers as it wishes). Then ensure the immediate children of `<Router>` have a prop named `path`.

A basic example of the `<Router>` component:

```
import React from "react"
import { Router } from "@reach/router"

export default function Routes() {
  return (
    <Router>
      <div path="/">I am the home!</div>
      <div path="/about">Here's a bit about me</div>
      <div path="/store">Buy my t-shirts!</div>
    </Router>
  )
}
```

Here's a more complex example of migrating a `<PrivateRoute>` component (used in store.gatsbyjs.org) from React Router to @reach/router.

```
 import * as React from 'react';
-import { Redirect, Route } from 'react-router-dom';
+import { Router, navigate } from '@reach/router';
 import { isAuthenticated } from '../../utils/auth';

-export default function PrivateRoute({ component: Component, ...rest }) {
-  return (
-    <Route
-      {...rest}
-      render={props =>
-        !isAuthenticated() ? (
-          // If we’re not logged in, redirect to the home page.
-          <Redirect to={{ pathname: '/login' }} />
-        ) : (
-          <Component {...props} />
-        )
-      }
-    />
-  );
-}
```

```
+export default function PrivateRoute({ component: Component, ...rest }) {
+  if (!isAuthenticated() && window.location.pathname !== `/login`) {
+    // If we’re not logged in, redirect to the home page.
+    navigate(`/app/login`);
+    return null;
+  }
+
+  return (
+    <Router>
+      <Component {...rest} />
+    </Router>
+  );
+};
```

Here are links to diffs for three sites with client routes upgraded to @reach/router:

- store.gatsbyjs.org
- client-only-routes
- simple-auth

APIs `onPreRouteUpdate` and `onRouteUpdate` no longer called with the route update action

React Router v4 would tell us the "action" (push/replace) that triggered the route transition. We passed this as one of the arguments along with `location` to plugins. @reach/router doesn't support this so we've removed it from the API calls.

Browser API `replaceRouterComponent` was removed

@reach/router doesn't allow you to swap out its history object like React Router. An API, `replaceRouterComponent`, was used with React Router for this behavior in Gatsby. This is now no longer needed, so we've removed this API.

We did, erroneously, suggest using this API for adding support for Redux, etc. where you need to wrap the root Gatsby component with your component.

If you were using `replaceRouterComponent` for this, you'll need to migrate to `wrapRootElement`:

```
import * as React from 'react'
import { Provider } from 'react-redux'
-import { Router } from 'react-router-dom'

-export const replaceRouterComponent = ({ history }) => {
+export const wrapRootElement = ({ element }) => {
-  const ConnectedRouterWrapper = ({ children }) => (
+  const ConnectedRootElement = (
    <Provider store={store}>
-      <Router history={history}>{children}</Router>
+      {element}
    </Provider>
  )

-  return ConnectedRouterWrapper
+   return ConnectedRootElement
}
```

Browser API `replaceHistory` was removed

Like with `replaceRouterComponent`, we no longer support custom histories. That is why we've also removed the `replaceHistory` API. The `replaceHistory()` method was used for tracking page views by registering listeners on route changes using `history.listen()`.

Now, to track page views, you can use the `onRouteUpdate` API to track pages changes.

Browser API `wrapRootComponent` was replaced with `wrapRootElement`

Use new `wrapRootElement` API: We now pass `component` Element instead of `Root` Component and expect that `wrapRootElement` will return Element and not Component. This change was needed to keep all wrapping APIs uniform.

```
-export const wrapRootComponent = ({ Root }) => {
+export const wrapRootElement = ({ element }) => {
-   const ConnectedRootComponent = () => (
+   const ConnectedRootElement = (
      <Provider store={store}>
-       <Root />
+       {element}
      </Provider>
    )
-   return ConnectedRootComponent
+   return ConnectedRootElement
}
```

Don't query nodes by ID

Source and transformer plugins now use UUIDs for IDs. If you used glob or regex to query nodes by id then you'll need to query something else.

Here's an example querying an image:

```
  query MyImageQuery {
    allImageSharp(filter: {
-     id: {regex: "/default.jpg/"}
+     fluid: {originalName: {regex: "/default.jpg/"}}
    }) {
      edges {
        node {
          id
          fluid(maxWidth: 660) {
            src
          }
        }
      }
    }
  }
```

See the Pull Request that implemented this change

Use `Query` in place of `RootQueryType`

We changed the GraphQL root type from `RootQueryType` to `Query`. This is only likely to impact you if you have top-level fragments in your GraphQL queries:

```
  query Blog {
    ...Sidebar
    blogPosts {
      title
      slug
    }
  }

- fragment Sidebar on RootQueryType {
+ fragment Sidebar on Query {
    siteDescription
  }
}
```

Typography.js Plugin Config Changes

If you use `gatsby-plugin-typography`, you now need to explicitly export `scale` and `rhythm` as named exports from your typography config module.

```
- const typography = new Typography();
- export default typography;

+ const typography = new Typography();
+ const { rhythm, scale } = typography;
+ export { rhythm, scale, typography as default };
```

Update CSS Modules class names that use dashes

If you use CSS Modules and have class names with dashes in them, you'll need to change how you access the class names from JavaScript.

This is because the `camelCase` option for CSS Modules has been changed from `false` to `dashesOnly`.

Here's an example with a class named `.my-class-name`:

```
import React from "react"
import * as myStyles from "./my.module.css"

export default function Component({ children }) (
-  <div className={myStyles['my-class-name']}>
+  <div className={myStyles.myClassName}>
```

```
    {children}
  </div>
)
```

The Gatsby v1 behavior can be restored by adjusting CSS Loader options.

For vanilla CSS without a preprocessor:

```
const cssLoaderRe = /\/css-loader\//
const targetFile = `.module.css`

const processRule = rule => {
  if (rule.oneOf) {
    return {
      ...rule,
      oneOf: rule.oneOf.map(processRule),
    }
  }

  if (!rule.test.test(targetFile)) {
    return rule
  }

  if (Array.isArray(rule.use)) {
    return {
      ...rule,
      use: rule.use.map(use => {
        if (!cssLoaderRe.test(use.loader)) {
          return use
        }

        // adjust css-loader options
        return {
          ...use,
          options: {
            ...use.options,
            camelCase: false,
          },
        }
      }),
    }
  }

  return rule
}

exports.onCreateWebpackConfig = ({ getConfig, actions }) => {
  const config = getConfig()

  const newConfig = {
    ...config,
    module: {
      ...config.module,
      rules: config.module.rules.map(processRule),
    },
  }
  actions.replaceWebpackConfig(newConfig)
}
```

If you're using a preprocessor, you can pass in CSS Loader options when configuring `gatsby-plugin-sass` or `gatsby-plugin-less` :

```
// in gatsby-config.js
plugins: [
  {
    resolve: `gatsby-plugin-sass`,
    options: {
      cssLoaderOptions: {
        camelCase: false,
      },
    },
  },
]
```

Update Jest configuration

If you were using Jest with Gatsby V1, you will need to make some updates to your configuration when upgrading to Gatsby V2. You can view the complete details of setting up your test environment on the Unit Testing page of the docs.

gatsby-image's `outerWrapperClassName` was removed

Because the outer wrapper `div` was removed, you can no longer use the `outerWrapperClassName` prop for styling your images. You should merge those styles into your wrapper's class.

```
<Img
  fluid={data.file.childImageSharp.fluid}
  className={styles.wrapper}
- outerWrapperClassName={styles.outerWrapper}
/>
```

If you have created any CSS styling rules referencing the `gatsby-image-outer-wrapper` class, you should merge those styles into the `gatsby-image-wrapper` class.

Resolving Deprecations

Import Link from Gatsby

All components and utility functions from `gatsby-link` are now exported from the `gatsby` package. So, you should import it directly from `gatsby` .

```
import React from "react"
- import Link from "gatsby-link"
+ import { Link } from "gatsby"

export default function Page(props) {
  return (
    <Link to="/">Home</Link>
  );
}
```

Furthermore, you can remove the package from the `package.json`.

```
"dependencies": {
  "gatsby": "latest",
  "gatsby-image": "latest",
  "gatsby-plugin-sharp": "latest",
- "gatsby-link": "^1.6.39"
}
```

Import GraphQL from Gatsby

The `graphql` tag function that Gatsby v1 auto-supports is deprecated in v2. Gatsby will throw deprecation warning unless you explicitly import it from the `gatsby` package.

```
import React from "react"
+ import { graphql } from "gatsby"

export default function Home({ data }) {
  return (
    <h1>{data.site.siteMetadata.title}</h1>
  );
}

export const query = graphql`
  query HomeQuery {
    site {
      siteMetadata {
        title
      }
    }
  }
`
```

> *There is a codemod that can automatically make this change to your projects. Check out the [`gatsby-codemods`] package for usage instructions.*

> *Note that if you are relying on the auto-import feature of WebStorm or VSCode, it may import `graphql` from `'graphql'` instead of `'gatsby'`. This will throw a bunch of errors around bad imports. Make sure `graphql` is always imported from `gatsby`.*

Rename `boundActionCreators` to `actions`

`boundActionCreators` is deprecated in v2. You can continue using it, but it's recommended that you rename it to `actions`.

Rename `pathContext` to `pageContext`

Like `boundActionCreators` above, `pathContext` is deprecated in favor of `pageContext`.

Rename responsive image queries

The `sizes` and `resolutions` queries are deprecated in v2. These queries have been renamed to `fluid` and `fixed` to make them easier to understand. You can continue using the deprecated query names, but it's recommended that you update them.

Update image query and fragment names:

```diff
import React from "react"
import { graphql } from "gatsby"

const Example = ({ data }) => {
  <div>
-    <Img sizes={data.foo.childImageSharp.sizes} />
-    <Img resolutions={data.bar.childImageSharp.resolutions} />
+    <Img fluid={data.foo.childImageSharp.fluid} />
+    <Img fixed={data.bar.childImageSharp.fixed} />
  </div>
}

export default Example

export const pageQuery = graphql`
  query IndexQuery {
    foo: file(relativePath: { regex: "/foo.jpg/" }) {
      childImageSharp {
-        sizes(maxWidth: 700) {
-          ...GatsbyImageSharpSizes_tracedSVG
+        fluid(maxWidth: 700) {
+          ...GatsbyImageSharpFluid_tracedSVG
        }
      }
    }
    bar: file(relativePath: { regex: "/bar.jpg/" }) {
      childImageSharp {
-        resolutions(width: 500) {
-          ...GatsbyImageSharpResolutions_withWebp
+        fixed(width: 500) {
+          ...GatsbyImageSharpFixed_withWebp
        }
      }
    }
  }
`
```

You can find further examples in the Gatsby Image docs.

Delete Nodes API Deprecated

`deleteNodes` is now deprecated, so instead you should write `nodes.forEach(n => deleteNode({ node: n }))`

Other Changes Worth Noting

Explicit query names no longer required

Gatsby v2 doesn't need explicit query names. You can skip them now:

```diff
export const query = graphql`
-  query ThisIsExplicitQueryName($slug: String!) {
+  query($slug: String!) {
    markdownRemark(fields: { slug: { eq: $slug } }) {
      html
      frontmatter {
        title
      }
    }
  }
```

You can also skip the `query` keyword if you don't use query variables:

```diff
export const query = graphql`
-  query ThisIsAnotherExplicitQueryName {
+  {
    site {
      siteMetadata {
        title
      }
    }
  }
```

This isn't a breaking change. Queries with explicit names will continue to work as they did in v1.

Remove inlined CSS in `html.js`

Gatsby v2 will automatically inline your CSS. You can remove any custom CSS inlining from your custom `html.js`. Unless you used it for anything else specifically, you can also remove `html.js` itself.

See an example in this PR that upgrades the `using-remark` site to Gatsby v2.

Remove explicit polyfills

If your Gatsby v1 site included any polyfills, you can remove them. Gatsby v2 ships with Babel 7 and is configured to automatically include polyfills for your code. See Gatsby's Babel docs for more details.

For Plugin Maintainers

In most cases you won't have to do anything to be v2 compatible, but there are a few things you can do to be certain your plugin will work well with v2 sites.

Setting the Proper Peer Dependencies

`gatsby` should be included under `peerDependencies` of your plugin and it should specify the proper versions of support.

```
"peerDependencies": {
-   "gatsby": "1"
+   "gatsby": ">=1"
}
```

Change `modifyBabelrc` to `onCreateBabelConfig`

We renamed `modifyBabelrc` to `onCreateBabelConfig` to bring it in line with the rest of Gatsby's API names.

Use `onCreateBabelConfig`:

```
- exports.modifyBabelrc = ({ babelrc }) => {
-   return {
-     ...babelrc,
-     plugins: babelrc.plugins.concat([`foo`]),
-   }
+ exports.onCreateBabelConfig = ({ actions }) => {
+   actions.setBabelPlugin({
+     name: `babel-plugin-foo`,
+   })
}
```

Note usage of the new `setBabelPlugin` action.

See Gatsby's Babel docs for more details about configuring Babel.

Change `modifyWebpackConfig` to `onCreateWebpackConfig`

We renamed `modifyWebpackConfig` to `onCreateWebpackConfig` to bring it in line with the rest of Gatsby's API names.

Use `onCreateWebpackConfig`:

```
- exports.modifyWebpackConfig = ({ config, stage }) => {
+ exports.onCreateWebpackConfig = ({ stage, actions }) => {
  switch (stage) {
    case `build-javascript`:
-       config.plugin(`Foo`, webpackFooPlugin, null)
-       break
-    }
-   return config
+       actions.setWebpackConfig({
+         plugins: [webpackFooPlugin],
+       })
+    }
}
```

Note usage of the new `setWebpackConfig` action.

See Gatsby's webpack docs for more details about configuring webpack.

createRemoteFileNode

The signature for using createRemoteFileNode changed in v2, it now expects a new parameter `createNodeId`.

See docs for `createRemoteFileNode`

Only allow defined keys on the node internal object

The node `internal` object isn't meant for adding node data. While Gatsby v1 allows this behavior we now validate against it for v2. Node data should be added as fields on the top-level node object.

Check the Node interface docs for allowed fields.

Import `graphql` types from `gatsby/graphql`

Import GraphQL types from `gatsby/graphql` to prevent `Schema must contain unique named types but contains multiple types named "<typename>"` errors. `gatsby/graphql` exports all builtin GraphQL types as well as `graphQLJSON` type.

```
-const { GraphQLString } = require(`graphql`)
+const { GraphQLString } = require(`gatsby/graphql`)
```

Add `gatsby-plugin-flow` if you are using Flowtype

We removed `@babel/preset-flow` from Gatsby's default Babel configuration to make it easier to allow users to choose their own transpiler. If your site has its own `.babelrc` that already includes the Flow preset, no changes are necessary. Otherwise, you should install `gatsby-plugin-flow`.

For Explorers

Starting a New Project with v2

Here's a brief section on starting a new project with Gatsby v2 instead of upgrading an existing project.

Start from scratch: If you're a *start from scratch* kind of person, you can install Gatsby and React like this: `npm install gatsby react react-dom`

Tutorial: If you'd like a step-by-step guide, follow the tutorial to get started with Gatsby v2.

Starters: If you'd rather use one of the official starters, install your favorite one with the Gatsby CLI.

`gatsby-starter-default` with v2:

```
gatsby new my-default-project https://github.com/gatsbyjs/gatsby-starter-default
```

`gatsby-starter-hello-world` with v2:

```
gatsby new my-hello-world https://github.com/gatsbyjs/gatsby-starter-hello-world
```

`gatsby-starter-blog` with v2:

```
gatsby new my-blog https://github.com/gatsbyjs/gatsby-starter-blog
```