Migrating from v2 to v3

Looking for the v2 docs?

> Have you run into something that's not covered here? Add your changes to GitHub!

Introduction

This is a reference for upgrading your site from Gatsby v2 to Gatsby v3. Since the last major release was in September 2018, Gatsby v3 includes a couple of breaking changes. If you're curious what's new, head over to the v3.0 release notes.

> If you want to start a new Gatsby v3 site, run `npm init gatsby` or `yarn create gatsby` in your terminal.

Table of Contents

Updating Your Dependencies

First, you need to update your dependencies.

Update Gatsby version

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

```
{
  "dependencies": {
    "gatsby": "^3.0.0"
  }
}
```

Or run

```
npm install gatsby@latest
```

Please note: If you use **npm 7** you'll want to use the `--legacy-peer-deps` option when following the instructions in this guide. For example, the above command would be:

```
npm install gatsby@latest --legacy-peer-deps
```

Update Gatsby related packages

Update your `package.json` to use the latest version 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 repository. If you're using community plugins, they might not be upgraded yet. Many plugins won't need updating so they may keep working (if not, please check their repository for the current status). You can run an npm script to see all outdated dependencies.

npm

```
npm outdated
```

Compare the "Wanted" and "Latest" versions and update their versions accordingly. For example, if you have this outdated version:

```
> npm outdated

Package                 Current   Wanted  Latest  Location
gatsby-plugin-sharp     2.14.1    2.14.1  3.0.0   test
```

Install the new package with:

```
npm install gatsby-plugin-sharp@latest
```

yarn

```
yarn upgrade-interactive --latest
```

You'll be given an overview of packages which to select to upgrade them to `latest`.

Updating community plugins Using community plugins, you might see warnings like these in your terminal:

```
warning Plugin gatsby-plugin-acme is not compatible with your gatsby version 3.0.0 - It requ
```

If you are using npm 7, the warning may instead be an error:

```
npm ERR! ERESOLVE unable to resolve dependency tree
```

This is because the plugin needs to update its `peerDependencies` to include the new version of Gatsby (see section for plugin maintainers). While this might indicate that the plugin has incompatibilities, in most cases they should continue

to work. When using npm 7, you can pass the `--legacy-peer-deps` to ignore the warning and install anyway. Please look for already opened issues or PRs on the plugin's repository to see the status. If you don't see any, help the maintainers by opening an issue or PR yourself! :)

Handling dependencies for plugins that are not yet updated Gatsby has an *amazing* ecosystem of plugins that make it easier to get up and running, and to incorporate various data sources and functionality into your Gatsby project. Part of that huge ecosystem includes dependency trees!

Depending on how the plugin authors have declared dependencies (e.g. marking a package as a dependency instead of a peerDependency) within those plugins, there could be a myriad of failures that arise. If you encounter any of these issues when migrating your project to Gatsby Version 3, we recommend that you use Yarn resolutions within your `package.json`.

Please note: If you rely on a plugin that is not found within the list of plugins within the Gatsby framework, you very well may need to use the following resolutions in the near term.

The specific resolutions we recommend at this time are found below:

```
{
  "resolutions": {
    "graphql": "^15.4.0",
    "graphql-compose": "^7.25.0",
    "webpack": "^5.24.2"
  }
}
```

Handling version mismatches

When upgrading an already existing project (that has an existing `node_modules` folder and `package-lock.json` file) you might run into version mismatches for your packages as npm/yarn don't resolve to the latest/correct version. An example would be a version mismatch of `webpack@4` and `webpack@5` that can throw an error like this:

```
Error: NormalModuleFactory.afterResolve (ReactRefreshPlugin) is no longer a waterfall hook,
```

An effective way to get around this issue is deleting `node_modules` and `package-lock.json` and then running `npm install` again. Alternatively, you can use `npm dedupe`.

Handling Breaking Changes

This section explains breaking changes that were made for Gatsby v3. Most, if not all, of those changes had a deprecation message in v2. In order to successfully update, you'll need to resolve these changes.

Minimal Node.js version 12.13.0

We are dropping support for Node 10 as it is approaching maintenance EOL date (2021-04-30). The new required version of Node is `12.13.0`. See the main changes in Node 12 release notes.

Check Node's releases document for version statuses.

webpack upgraded from version 4 to version 5

We tried our best to mitigate as much of the breaking change as we could. Some are sadly inevitable. We suggest looking at the official webpack 5 blog post to get a comprehensive list of what changed.

If you hit any problems along the way, make sure the Gatsby plugin or webpack plugin supports version 5.

ESLint upgraded from version 6 to version 7

If you're using Gatsby's default ESLint rules (no custom `eslintrc` file), you shouldn't notice any issues. If you do have a custom ESLint config, make sure to read the ESLint 6 to 7 migration guide

`src/api` is a reserved directory now

With the release of Gatsby 3.7 we introduced Functions. With this any JavaScript or TypeScript files inside `src/api/*` are mapped to function routes like files in `src/pages/*` become pages. This also means that if you already have an existing `src/api` folder you'll need to rename it to something else as it's a reserved directory now.

Gatsby's `Link` component

The APIs `push`, `replace` & `navigateTo` in `gatsby-link` (an internal package) were deprecated in v2 and now with v3 completely removed. Please use `navigate` instead.

```
import React from "react"
- import { navigateTo, push, replace } from "gatsby"
+ import { navigate } from "gatsby"

const Form = () => (
  <form
    onSubmit={event => {
      event.preventDefault()

-     navigateTo("/form-submitted/") // or push() or replace()
+     navigate("/form-submitted/")
    }}
```

```
  >
)
```

Removal of `__experimentalThemes`

The deprecated `__experimentalThemes` key inside `gatsby-config.js` was removed. You'll need to define your Gatsby themes inside the `plugins` array instead.

```
module.exports = {
- __experimentalThemes: ["gatsby-theme-blog"]
+ plugins: ["gatsby-theme-blog"]
}
```

Removal of `pathContext`

The deprecated API `pathContext` was removed. You need to rename instances of it to `pageContext`. For example, if you passed information inside your `gatsby-node.js` and accessed it in your page:

```
import React from "react"

- const ContextPage = ({ pathContext }) => (
+ const ContextPage = ({ pageContext }) => (
  <main>
    <h1>Hello from a page that uses the old pathContext</h1>
    <p>It was deprecated in favor of pageContext</p>
-   <p>{pathContext.foo}</p>
+   <p>{pageContext.foo}</p>
  </main>
)

export default ContextPage
```

Removal of `boundActionCreators`

The deprecated API `boundActionCreators` was removed. Please rename its instances to `actions` to keep the same behavior. For example, in your `gatsby-node.js` file:

```
exports.createPages = (gatsbyArgs, pluginArgs) => {
- const { boundActionCreators } = gatsbyArgs
+ const { actions } = gatsbyArgs
}
```

Removal of `deleteNodes`

The deprecated API `deleteNodes` was removed. Please iterate over the `nodes` instead and call `deleteNode`:

```
const nodes = ["an-array-of-nodes"]
- deleteNodes(nodes)
+ nodes.forEach(node => deleteNode(node))
```

Removal of `fieldName` & `fieldValue` from `createNodeField`

The arguments `fieldName` and `fieldValue` were removed from the `createNodeField` API. Please use `name` and `value` instead.

```
exports.onCreateNode = ({ node, actions }) => {
  const { createNodeField } = actions

  createNodeField({
    node,
-   fieldName: "slug",
-   fieldValue: "my-custom-slug",
+   name: "slug",
+   value: "my-custom-slug",
  })
}
```

Removal of `hasNodeChanged` from public API surface

This API is no longer necessary, as there is an internal check for whether or not a node has changed.

Removal of `sizes` & `resolutions` for image queries

The `sizes` and `resolutions` queries were deprecated in v2 in favor of `fluid` and `fixed`.

While `fluid`, `fixed`, and `gatsby-image` will continue to work in v3, we highly recommend migrating to the new `gatsby-plugin-image`. Read the Migrating from `gatsby-image` to `gatsby-plugin-image` guide to learn more about its benefits and how to use it.

```
import React from "react"
import { graphql } from "gatsby"
import Img from "gatsby-image"

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
        }
      }
    }
  }
`
```

Calling `touchNode` with the node id

Calling `touchNode` with a string (the node id) was deprecated in Gatsby v2. Pass the full `node` to `touchNode` now.

```
exports.sourceNodes = ({ actions, getNodesByType }) => {
  const { touchNode } = actions

- getNodesByType("YourSourceType").forEach(node => touchNode(node.id))
+ getNodesByType("YourSourceType").forEach(node => touchNode(node))
}
```

Calling `deleteNode` with the node id

Calling `deleteNode` with a string (the node id) was deprecated in Gatsby v2. Pass the full `node` to `deleteNode` now.

```
exports.onCreateNode = ({ actions, node }) => {
  const { deleteNode } = actions

- deleteNode(node.id)
+ deleteNode(node)
}
```

Removal of three `gatsby-browser` APIs

A couple of `gatsby-browser` APIs were removed. In the list below you can find the old APIs and their replacements:

- `getResourcesForPathnameSync` => `loadPageSync`
- `getResourcesForPathname` => `loadPage`
- `replaceComponentRenderer` => `wrapPageElement`

Using a global `graphql` tag for queries

Until now you were able to use the `graphql` tag for queries without explicitly importing it from Gatsby. You now have to import it: `import { graphql } from 'gatsby'`

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

const Page = ({ data }) => (
  <div>Show my data: {JSON.stringify(data, null, 2)}</div>
)

export default Page

export const query = graphql`
  {
    site {
      siteMetadata {
        description
      }
    }
  }
`
```

CSS Modules are imported as ES Modules

The web moves forward and so do we. ES Modules allow us to better treeshake and generate smaller files. From now on you'll need to import cssModules as: `import { box } from './mystyles.module.css'`

```
import React from "react"
- import styles from './mystyles.module.css'
+ import { box } from './mystyles.module.css'

const Box = ({ children }) => (
-  <div className={styles.box}>{children}</div>
+  <div className={box}>{children}</div>
)

export default Box
```

You can also import all styles using the `import * as styles` syntax e.g. `import * as styles from './mystyles.module.css'`. However, this won't allow webpack to treeshake your styles so we discourage you from using this syntax.

Migrating all your CSS could be painful or you're relying on third-party packages that require you to use CommonJS. You can work around this issue for Sass, Less, Stylus & regular CSS modules using respective plugins. If you're using regular CSS modules, please install gatsby-plugin-postcss to override the defaults.

This example covers Sass. The other plugins share the same `cssLoaderOptions` property.

```
module.exports = {
  plugins: [
-    `gatsby-plugin-sass`,
+    {
+      resolve: `gatsby-plugin-sass`,
+      options: {
+       cssLoaderOptions: {
+         esModule: false,
+         modules: {
+           namedExport: false,
+         },
+       },
+     },
+    }
  ]
}
```

File assets (fonts, pdfs, . . .) are imported as ES Modules

Assets are handled as ES Modules. Make sure to switch your require functions into imports.

```
import React from "react"
import { Helmet } from "react-helmet";
+ import myFont from '../assets/fonts/myfont.woff2'
```

```
const Layout = ({ children }) => (
  <div>
    <Helmet>
-      <link rel="preload" href={require('../assets/fonts/myfont.woff2')} as="fonts/woff2" c
+      <link rel="preload" href={myFont} as="fonts/woff2" crossOrigin="anonymous" type="font
    </Helmet>
    {children}
  </div>
)

export default Layout
```

If you're using `require with expression` or `require.context` (which is not recommended), you'll have to append `.default` to your require statement to make it work.

```
import React from "react"
import { Helmet } from "react-helmet";

const Layout = ({ children, font }) => (
  <div>
    <Helmet>
-      <link rel="preload" href={require('../assets/fonts/' + font + '.woff2')} as="fonts/wo
+      <link rel="preload" href={require('../assets/fonts/' + font + '.woff2').default} as="
    </Helmet>
    {children}
  </div>
)

export default Layout
```

webpack 5 node configuration changed (node.fs, node.path, ...)

Some components need you to patch/disable node APIs in the browser, like `path` or `fs`. webpack removed these automatic polyfills. You now have to manually set them in your configurations:

```
exports.onCreateWebpackConfig = ({ actions }) => {
  actions.setWebpackConfig({
-    node: {
-      fs: "empty",
-      path: "mock",
-    },
+    resolve: {
+       alias: {
+          path: require.resolve("path-browserify")
```

```
+       },
+       fallback: {
+         fs: false,
+       }
+    }
  })
}
```

If it's still not resolved, the error message should guide you on what else you need to add to your webpack config.

process is not defined A common error is `process is not defined`. webpack 4 polyfilled process automatically in the browser, but with v5 it's not the case anymore.

If you're using `process.browser` in your components, you should switch to a window is not undefined check.

```
import React from "react"

const Base64 = ({ text }) => {
  let base64;
-  if (process.browser) {
+  if (typeof window !== "undefined") {
    base64 = btoa(text)
  } else {
    base64 = Buffer.from(text).toString('base64')
  }

  return (
    <div>
      {base64}
    </div>
  )
}

export default Base64
```

If you're using any other process properties, you want to polyfill process.

1. Install `process` library - `npm install process`
2. Configure webpack to use the process polyfill.

```
exports.onCreateWebpackConfig = ({ actions, stage, plugins }) => {
  if (stage === 'build-javascript' || stage === 'develop') {
    actions.setWebpackConfig({
      plugins: [
        plugins.provide({ process: 'process/browser' })
```

```
      ]
    })
  }
}
```

GraphQL: character escape sequences in `regex` filter

In v2, backslashes in `regex` filters of GraphQL queries had to be escaped *twice*, so `/\w+/` needed to be written as `"/\\\\w+/"`.

In v3, you only need to escape once:

```
const query = {
  allFile(
    filter: {
-      relativePath: { regex: "/\\\\.png/" }
+      relativePath: { regex: "/\\.png/" }
    }
  ) {
    nodes {
      relativePath
    }
  }
}
```

GraphQL: `__typename` field is no longer added automatically

In v2, the `__typename` field used to be added implicitly when querying for a field of abstract type (interface or union). In v3, `__typename` has to be added explicitly in your query:

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

const Page = ({ data }) => {
  if (data.foo.someUnion.__typename === `A`) {
    return data.foo.someUnion.a
  }
  if (data.foo.someUnion.__typename === `B`) {
    return data.foo.someUnion.b
  }
  return null
}

export default Page

export const query = graphql`
  {
```

```
    foo {
      someUnion {
+       __typename
        ... on A { a }
        ... on B { b }
      }
    }
  }
`
```

Schema Customization: Add explicit `childOf` extension to types with disabled inference

Imagine you have node type `Foo` that has several child nodes of type `Bar` (so you expect field `Foo.childBar` to exist). In Gatsby v2 this field was added automatically even if inference was disabled for type `Foo`.

In Gatsby v3 you must declare a parent-child relationship explicitly for this case:

```
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
    type Foo implemenst Node @dontInfer {
       id: ID!
    }
-   type Bar implements Node {
+   type Bar implements Node @childOf(types: ["Foo"]) {
      id: ID!
    }
  `)
}
```

To make upgrading easier, check the CLI output of your site on the latest v2 and follow the suggestions when you see a warning like this:

```
warning Deprecation warning: In Gatsby v3 fields `Foo.childBar` and `Foo.childrenBar`
will not be added automatically because type `Bar` does not explicitly list type `Foo`
in `childOf` extension.

Add the following type definition to fix this:

  type Bar implements Node @childOf(types: ["Foo"]) {
    id: ID!
  }

https://www.gatsbyjs.com/docs/actions/#createTypes
```

If you don't see any warnings, you are safe to upgrade to v3.

If this warning is displayed for a type defined by some plugin, open an issue in the plugin repo with a suggestion to upgrade (and a link to this guide).

You can still fix those warnings temporarily in your site's `gatsby-node.js` file until it is fixed in the plugin.

Related docs:

- childOf directive
- child/parent fields
- schema generation

Schema Customization: Extensions must be set explicitly

Starting with v3, whenever you define a field of complex type, you must also assign the corresponding extension (or a custom resolver):

```
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
    type MyType {
-     date: Date
+     date: Date @dateformat
-     image: File
+     image: File @fileByRelativePath
-     authorByEmail: Author
+     authorByEmail: Author @link
    }
  `)
}
```

In Gatsby v2, we add those extensions for you automatically but display a deprecation warning.

To make upgrading easier, when you see a warning like the one below, check the CLI output of your site on the latest v2 and follow the suggestions provided.

```
warning Deprecation warning: adding inferred extension `link` for field Foo.bar
In Gatsby v3, only fields with an explicit directive/extension will be resolved correctly.
Add the following type definition to fix this:

  type Foo implements Node {
    bar: [Bar] @link(by: "id", from: "bar___NODE")
  }

https://www.gatsbyjs.com/docs/actions/#createTypes
```

If this warning is displayed for a type defined by some plugin, open an issue in the plugin repo with a suggestion to upgrade (and a link to this guide).

You can still fix those warnings temporarily in your site's `gatsby-node.js` until it is fixed in the plugin.

If you don't see any warnings, you are safe to upgrade to v3. Read more about custom extensions in this blog post.

Schema Customization: Removed `noDefaultResolvers` argument from inference directives

Search for `noDefaultResolvers` entries and remove them:

```
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
-   type Foo implements Node @infer(noDefaultResolvers: true)
+   type Foo implements Node @infer
    {
      id: ID!
    }

-   type Bar implements Node @dontInfer(noDefaultResolvers: true)
+   type Foo implements Node @dontInfer
    {
      id: ID!
    }

  `)
}
```

Deprecation announcement for `noDefaultResolvers`.

Schema Customization: Remove `many` argument from `childOf` directive

The `many` argument is no longer needed for the `childOf` directive in Gatsby v3:

```
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
-   type Foo implements Node @childOf(types: ["Bar"], many: true)
+   type Foo implements Node @childOf(types: ["Bar"])
    {
      id: ID!
    }
  `)
}
```

Schema Customization: Consistent return for `nodeModel.runQuery`

For Gatsby v2, `nodeModel.runQuery` with `firstOnly: false` returns `null` when nothing is found. In v3 it returns an empty array instead.

To upgrade, find all occurrences of `runQuery` (with `firstOnly: false` or not set) and make sure checks for emptiness are correct:

```
exports.createResolvers = ({ createResolvers }) => {
  const resolvers = {
    Foo: {
      bars: {
        resolve(source, args, context, info) {
          const result = context.nodeModel.runQuery({
            query: {/* */},
            type: "Bar",
            firstOnly: false,
          })
-         if (result === null) {
+         if (result.length === 0) {
            throw new Error("Not found!")
          }
          return result
        },
      },
    },
  }
  createResolvers(resolvers)
}
```

Note: When using argument `firstOnly: true` the returned value is `object` or `null`. So do not confuse those two cases.

Future Breaking Changes

This section explains deprecations that were made for Gatsby v3. These old behaviors will be removed in v4, at which point they will no longer work. For now, you can still use the old behaviors in v3, but we recommend updating to the new signatures to make future updates easier.

`touchNode`

For Gatsby v2 the `touchNode` API accepted `nodeId` as a named argument. This now has been deprecated in favor of passing the full `node` to the function.

```
exports.sourceNodes = ({ actions, getNodesByType }) => {
  const { touchNode } = actions
```

```
- getNodesByType("YourSourceType").forEach(node => touchNode({ nodeId: node.id }))
+ getNodesByType("YourSourceType").forEach(node => touchNode(node))
}
```

In case you only have an ID at hand (e.g. getting it from cache or as `__NODE`), you can use the `getNode()` API:

```
exports.sourceNodes = async ({ actions, getNode, getNodesByType, cache }) => {
  const { touchNode } = actions
  const myNodeId = await cache.get("some-key")

  touchNode(getNode(myNodeId))
}
```

deleteNode

For Gatsby v2, the `deleteNode` API accepted `node` as a named argument. This now has been deprecated in favor of passing the full `node` to the function.

```
exports.onCreateNode = ({ actions, node }) => {
  const { deleteNode } = actions

- deleteNode({ node })
+ deleteNode(node)
}
```

@nodeInterface

For Gatsby v2, `@nodeInterface` was the recommended way to implement queryable interfaces. Now it is deprecated in favor of interface inheritance:

```
exports.createSchemaCustomization = function createSchemaCustomization({ actions }) {
  const { createTypes } = actions
  createTypes(`
-   interface Foo @nodeInterface
+   interface Foo implements Node
    {
      id: ID!
    }
  `)
}
```

JSON imports: follow the JSON modules web spec

JSON modules are coming to the web. JSON modules only allow you to import the default export and no sub-properties. If you do import properties, you'll get a warning along these lines:

```
Should not import the named export 'myProp' from default-exporting module (only default expo
```

```
import React from "react"
- import { items } from "../data/navigation.json"
+ import navigationData from "../data/navigation.json"

const Navigation = () => (
  <nav>
    <ul>
-      {items.map(item => {
-        return <li><a href={item.to}>{item.text}</a></li>
-      })}
+      {navigationData.items.map(item => {
+        return <li><a href={item.to}>{item.text}</a></li>
+      })}
    </ul>
  </nav>
)

export default Navigation
```

webpack deprecation messages

When running community Gatsby plugins, you might see `[DEP_WEBPACK]` messages popup during the "Building JavaScript" or the "Building SSR bundle" phase. These often mean that the plugin is not compatible with webpack 5 yet. Contact the Gatsby plugin author or the webpack plugin author to flag this issue. Most of the time Gatsby will build fine, however there are cases that it won't and the reasons why could be cryptic.

Using `fs` in SSR

Gatsby v3 introduces incremental builds for HTML generation. For this feature to work correctly, Gatsby needs to track all inputs used to generate HTML file. Arbitrary code execution in `gatsby-ssr.js` files allow usage of `fs` module, which is marked as unsafe and results in disabling of this feature. To migrate, you can use `import` instead of `fs`:

```
import * as React from "react"
-import * as fs from "fs"
-import * as path from "path"
+import stylesToInline from "!!raw-loader!/some-auto-generated.css"

export function onRenderBody({ setHeadComponents }) {
-  const stylesToInline = fs.readFileSync(path.join(process.cwd(), `some-auto-generated.css`
  setHeadComponents(
    <style
      dangerouslySetInnerHTML={{
        __html: stylesToInline,
```

```
      }}
    />
  )
}
```

For Plugin Maintainers

In most cases, you won't have to do anything to be v3 compatible. But one thing you can do to be certain your plugin won't throw any warnings or errors is to set the proper peer dependencies.

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

```
{
  "peerDependencies": {
-   "gatsby": "^2.32.0",
+   "gatsby": "^3.0.0",
  }
}
```

If your plugin supports both versions:

```
{
  "peerDependencies": {
-   "gatsby": "^2.32.0",
+   "gatsby": "^2.32.0 || ^3.0.0",
  }
}
```

Known Issues

This section is a work in progress and will be expanded when necessary. It's a list of known issues you might run into while upgrading Gatsby to v3 and how to solve them.

`reach-router`

We vendored reach-router to make it work for React 17. We added a webpack alias so that you can continue using it as usual. However, you might run into an error like this after upgrading:

```
Generating development JavaScript bundle failed

Can't resolve '@gatsbyjs/reach-router/lib/utils' in '/c/Users/xxx/test/node_modules/gatsby-l

If you're trying to use a package make sure that '@gatsbyjs/reach-router/lib/utils' is insta
is correct.
```

`File: node_modules/gatsby-link/index.js:24:13`

To resolve the error above, make sure that you have updated all dependencies. It's also possible that you have an outdated `.cache` folder around. Run `gatsby clean` to remove the outdated cache.

In some situations the webpack alias will be ignored, so you will need to add your own alias. The most common example is in Jest tests. For these, you should add the following to your Jest config:

Configuring using a `jest.config.js` file:

```
module.exports = {
  moduleNameMapper: {
    "^@reach/router(.*)": "<rootDir>/node_modules/@gatsbyjs/reach-router$1",
  },
}
```

Configuring using `package.json`:

```
{
  "jest": {
    "moduleNameMapper": {
      "^@reach/router(.*)": "<rootDir>/node_modules/@gatsbyjs/reach-router$1"
    }
  }
}
```

webpack EACCES

You might see errors like these when using Windows or WSL:

```
Watchpack Error (initial scan): Error: EACCES: permission denied, lstat '/c/DumpStack.log.tm
Watchpack Error (initial scan): Error: EACCES: permission denied, lstat '/c/hiberfil.sys'
Watchpack Error (initial scan): Error: EACCES: permission denied, lstat '/c/pagefile.sys'
Watchpack Error (initial scan): Error: EACCES: permission denied, lstat '/c/swapfile.sys'
```

Gatsby will continue to work. Please track the upstream issue to see how and when this will be fixed.

yarn workspaces

Workspaces and their hoisting of dependencies can cause you troubles if you incrementally want to update a package. For example, if you use `gatsby-plugin-emotion` in multiple packages but only update its version in one, you might end up with multiple versions inside your project. Run `yarn why package-name` (in this example `yarn why gatsby-plugin-emotion`) to check if different versions are installed.

We recommend updating all dependencies at once and re-checking it with `yarn why package-name`. You should only see one version found now.