This guide focuses on the parts of Gatsby that are applicable to a static website. For a more comprehensive walk through Gatsby's features check out the Gatsby tutorial. By following the example on this page, you will complete the key stages of porting an HTML website to Gatsby and establish your Gatsby development workflow.

**Note:** This guide can also be used to migrate a section of a site, to be served next to existing files. Pay extra attention to the section on hosting for guidance.

## Getting Started

Here is the structure of an example static HTML/CSS website that this guide will walk through porting:

```
website-domain
  ├── assets
  │   ├── favicon.ico
  │   ├── person.png
  │   ├── normalize.css
  │   └── style.css
  ├── index.html
  ├── 404.html
  ├── about.html
  ├── contact.html
  ├── services
  │   ├── index.html
  │   ├── growing.html
  │   ├── cleaning.html
  │   └── shrinking.html
  └── who
      ├── index.html
      ├── ellla-arborist.html
      ├── marin-leafer.html
      └── sam-surgeon.html
```

### Assumptions

The example site uses global CSS files ( `style.css` and `normalize.css` ); styling structures like Sass architectures or CSS-in-JS can be accommodated but will not be covered here.

No client-side JavaScript (e.g jQuery etc.) is on the example site. If your site includes client-side JavaScript libraries and functionality, Gatsby may conflict with it at build time if not handled or removed when porting. Learn more about Debugging HTML Builds.

### Development environment

Gatsby generates websites and web applications for production through a compilation and build process, and it also has tools optimized for local development. To set up the Gatsby CLI and development environment (if you haven't already) check out Part Zero of the Gatsby tutorial.

### Gatsby Project

Now that you are set up, you can use the Gatsby and npm CLI tools in your terminal to get this site ported! Make a new project using the Gatsby hello world starter with the following command:

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

You should now have a folder called `gatsby-site` containing a basic Gatsby application. Open the new folder in your code editor and `cd` (change directory) into the folder in your terminal to continue:

```
cd gatsby-site
```

The `/src` folder contains most of the front-end code for the Gatsby site. In the Gatsby build process, every component file in the `/src/pages` folder will automatically create an HTML page. In your new Gatsby application, the only page created is from the index page component in `/src/pages/index.js`:

```
import React from "react"

export default function Home() {
  return <div>Hello world!</div>
}
```

Run the development server with `gatsby develop` in the command line to see the website in your browser.

```
gatsby develop
```

You can now visit the page running in your browser at `http://localhost:8000`. Hello Gatsby! 👋

## Porting index.html

Here is `/index.html` from the example site structure above:

```
<html lang="en">
  <head>
    <title>Taylor's Tidy Trees</title>
    <link href="/assets/favicon.ico" rel="shortcut icon" type="image/x-icon" />
    <link rel="stylesheet" type="text/css" href="/assets/normalize.css" />
    <link rel="stylesheet" type="text/css" href="/assets/style.css" />
  </head>
  <body>
    <header>
      <a href="/" class="brand-color logo-text">Taylor's Tidy Trees</a>
      <nav>
        <ul>
          <li><a href="/about.html">About</a></li>
          <li><a href="/services/index.html">Services</a></li>
          <li><a href="/who/index.html">Who We Are</a></li>
          <li><a href="/contact.html">Contact</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <h1>Welcome To Taylor's Tidy Trees</h1>
```

```
      <h2>We care about trees of all kinds!</h2>
    </main>
  </body>
</html>
```

In the following sections, you'll convert this block of HTML into its equivalent code in Gatsby.

### Assets

In the `/static` folder of your new Gatsby project, you can see a `favicon.ico` file, this is the Gatsby favicon! All files within `/static` will be served at the root of the app, you can put yours in there and watch it show up in the browser. The other image file in this example, `person.png`, should also be moved to the `/static` folder for later use.

Gatsby's bundling system negates the need for manual CSS `<link>` tags for local CSS. Within the `/src/pages/index.js` file, before the page component is defined, you can add a line for importing each CSS file. Gatsby will then efficiently deliver the CSS with your site.

Create a folder at `/src/styles` in the Gatsby project. All the CSS files for the project can be moved into this new folder. Add an `import` line for each CSS file to the Gatsby home page:

```
import React from "react"

import "../styles/normalize.css" // highlight-line
import "../styles/style.css" // highlight-line

export default function Home() {
  return <div>Hello world!</div>
}
```

### Head elements

You might have noticed that the component in `/src/pages/index.js` doesn't include `<html>`, `<head>` or `<body>`. Gatsby makes a default HTML structure for each page and places the output from `/src/pages/index.js` into its body. More `<head>` child elements and HTML attributes are added to the output page with a module called React Helmet. React Helmet is added to a Gatsby project in the command line with npm and then to the Gatsby config file:

```
npm install react-helmet gatsby-plugin-react-helmet
```

Gatsby projects have a config file at `/gatsby-config.js` where site metadata and options can be specified and plugins added. Add a plugin line with `gatsby-plugin-react-helmet` to your config file:

```
/**
 * Configure your Gatsby site with this file.
 *
 * See: https://www.gatsbyjs.com/docs/reference/config-files/gatsby-config/
 */

module.exports = {
```

```
  plugins: ["gatsby-plugin-react-helmet"], // highlight-line
}
```

Now you can import the `<Helmet>` component to the `index.js` file and place `<header>` & `<main>` elements for the existing HTML. Copy over the contents of the `<head>` tag: you'll no longer need the `<link>` tags for the CSS files. The Gatsby components must have a single root parent in their code structure so one technique is to add a [React Fragment component] around them:

```
import React from "react"
import { Helmet } from "react-helmet" // highlight-line

import "../styles/normalize.css"
import "../styles/style.css"

export default function Home() {
  return (
    <>
      <Helmet>
        {/* highlight-start */}
        <title>Taylor's Tidy Trees</title>
        <link href="/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        {/* highlight-end */}
      </Helmet>
      <header></header>
      <main>
        <div>Hello world!</div>
      </main>
    </>
  )
}
```

Note the mix of components and native HTML elements in the React markup here: this is the [JSX] templating language, which Gatsby compiles into HTML that browsers can parse and render to users. Further sections of this guide will explain it even more.

### Site navigation

Gatsby itself provides a number of core building blocks: `<Link>` is one of them. The `<Link>` component is imported at the top of the file to use in place of `<a>` tags for internal links, with a `to` prop instead of the `href` attribute. When the site builds, `<Link>` produces native HTML anchors with added performance optimizations like prefetching page content before a user activates a link.

Copy over the `<header>` element contents, changing `<a>` elements to `<Link>` components:

```
import React from "react"
import { Helmet } from "react-helmet"
import { Link } from "gatsby" // highlight-line

import "../styles/normalize.css"
import "../styles/style.css"
```

```
export default function Home() {
  return (
    <>
      <Helmet>
        <title>Taylor's Tidy Trees</title>
        <link href="/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        <link rel="stylesheet" type="text/css" href="/assets/normalize.css" />
        <link rel="stylesheet" type="text/css" href="/assets/style.css" />
      </Helmet>
      <header>
        {/* highlight-start */}
        <Link to="/" className="brand-color logo-text">
          Taylor's Tidy Trees
        </Link>
        <nav>
          <ul>
            <li>
              <Link to="/about">About</Link>
            </li>
            <li>
              <Link to="/services">Services</Link>
            </li>
            <li>
              <Link to="/who">Who We Are</Link>
            </li>
            <li>
              <Link to="/contact">Contact</Link>
            </li>
          </ul>
        </nav>
        {/* highlight-end */}
      </header>
      <main>
        <div>Hello world!</div>
      </main>
    </>
  )
}
```

### Page content

The contents of the `<main>` tag can be copied over from `index.html` to your new `index.js` file mostly unchanged. Your pasted HTML needs a small change to be valid: `class` attributes must be renamed to `className` for usage with React, as `class` is a reserved word in JavaScript.

Opening the site in a browser again at `http://localhost:8000`, you should have a visually complete home page! You'll make the links functional next by porting more pages. Before that, this guide will briefly explore how HTML and JavaScript combine as JSX in a Gatsby application.

### HTML from JavaScript

The code for Gatsby pages looks like a hybrid of JavaScript and HTML. The code for each page is typically a JavaScript function describing a block of HTML given a set of inputs, or "props". Gatsby runs each page's JavaScript function during the build process to produce a static HTML file.

The appearance of a Gatsby component depends on how dynamic the content and behavior is. The code for a very static page will include mostly all HTML markup wrapped in a bit of JavaScript for Gatsby to assemble. The code for a component with props (a.k.a. "inputs"), and logic applied to those props, will interweave more JavaScript through JSX: examples could include data sourced with GraphQL or imported from a file to produce dynamic markup, such as a list of related links.

This guide will stay on the HTML side of the balance to suit a more static site. Using Gatsby to arrange the necessary client-side JavaScript with React early can open many future possibilities though. While Gatsby produces static pages from your components, it can also deliver dynamic client-side JavaScript after the page loads and the site hydrates into a full React web application.

## Porting pages

### Porting a sub-index page

There are 4 pages in the `/who` section of Taylor's Tidy Trees for members of Taylor's tree team. Here is the index page:

```
<html lang="en">
  <head>
    <title>Taylor's Tidy Trees - Who We Are</title>
    <link href="/assets/favicon.ico" rel="shortcut icon" type="image/x-icon" />
    <link rel="stylesheet" type="text/css" href="/assets/normalize.css" />
    <link rel="stylesheet" type="text/css" href="/assets/style.css" />
  </head>
  <body>
    <header>
      <a href="/" class="brand-color logo-text">Taylor's Tidy Trees</a>
      <nav>
        <ul>
          <li><a href="/about.html">About</a></li>
          <li><a href="/services/index.html">Services</a></li>
          <li><a href="/index.html">Who We Are</a></li>
          <li><a href="/contact.html">Contact</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <h1>Who We Are</h1>
      <h2>These are our staff:</h2>
      <ul>
        <li><a href="/who/ella-arborist.html">Ella (Arborist)</a></li>
        <li><a href="/who/sam-surgeon.html">Sam (Tree Surgeon)</a></li>
        <li><a href="/who/marin-leafer.html">Marin (Leafer)</a></li>
      </ul>
    </main>
  </body>
</html>
```

### Layout component

The foundational building block for building and styling pages in Gatsby is the `<Layout>` component. The `<Layout>` component wraps around page content, providing the common structure that appears on all pages. Looking at the `/index.html` and `/who/index.html` you can see that most of the page is identical. Other than the title of the page, everything except for the contents of the main block is repeated.

Create a folder inside `/src`, next to `/src/pages` called `components`. Inside `components` make a file called `Layout.js`.

Like in `/src/pages/index.js` the file exports a JavaScript function that returns an HTML-like JSX structure, but this time the function takes an argument. The first argument provided to a component function is always an object for the props. On the props object, the children of the component are available to be passed in. Within the JSX markup, the curly braces wrap a JavaScript expression whose result will be placed there. In this case it is an expression that results in the contents of the `children` variable.

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

export default function Layout({ children }) {
  return (
    <>
      <Helmet></Helmet>
      <header></header>
      <main>{children}</main>
    </>
  )
}
```

The common elements between the `/index.html` and `/who/index.html` files can now copied from `/src/index.js` into the `<Layout>` component. A second prop is also added here and used in a JavaScript expression in the `<title>` element. The new expression results in a dynamic version of the title, depending on what is passed as the `breadcrumbs` prop. If the `breadcrumbs` prop is provided, it is joined and added to the end of the base title, `Taylor's Tidy Trees`. You'll see the prop in use later on when porting the `/who/index.html` page.

```
import React from "react"
import { Helmet } from "react-helmet"
import { Link } from "gatsby"

import "../styles/normalize.css"
import "../styles/style.css"

export default function Layout({ children, breadcrumbs }) {
  return (
    <>
      <Helmet>
        <title>
          Taylor's Tidy Trees
```

```
          {breadcrumbs ? ` - ${breadcrumbs.join(" - ")}` : ``}
        </title>
        <link href="/favicon.ico" rel="shortcut icon" type="image/x-icon" />
      </Helmet>
      <header>
        <Link to="/" className="brand-color logo-text">
          Taylor's Tidy Trees
        </Link>
        <nav>
          <ul>
            <li>
              <Link to="/about.html">About</Link>
            </li>
            <li>
              <Link to="/services/index.html">Services</Link>
            </li>
            <li>
              <Link to="/who/index.html">Who We Are</Link>
            </li>
            <li>
              <Link to="/contact.html">Contact</Link>
            </li>
          </ul>
        </nav>
      </header>
      <main>{children}</main>
    </>
  )
}
```

You can now use the `<Layout>` component to create a `/src/who/index.js` page file:

```
import React from "react"
import Layout from "../components/Layout"
import { Link } from "gatsby"

export default function Who() {
  return (
    <Layout breadcrumbs={["Who We Are"]}>
      <h1>Who We Are</h1>
      <h2>These are our staff:</h2>
      <ul>
        <li>
          <Link to="/who/ella-arborist">Ella (Arborist)</Link>
        </li>
        <li>
          <Link to="/who/sam-surgeon">Sam (Tree Surgeon)</Link>
        </li>
        <li>
          <Link to="/who/marin-leafer">Marin (Leafer)</Link>
        </li>
```

```
      </ul>
    </Layout>
  )
}
```

The `Who We Are` link in `index.js` should now work! Now use the `<Layout>` component in the `index.js` page file too:

```
import React from "react"
import Layout from "../components/Layout" // highlight-line

export default function Home() {
  return (
    {/* highlight-start */}
    <Layout>
      <h1>Welcome To Taylor's Tidy Trees</h1>
      <h2>We care about trees of all kinds!</h2>
    </Layout>
    {/* highlight-end */}
  );
}
```

Have a check that the `Who We Are` link is still working. If not, check that the content is wrapped correctly with the `<Layout>` component as shown above.

### Porting other pages

Now it's time for the work to really pay off! Ella's page is a matter of using your `<Layout>` component again and copying in the main content. Don't forget to change `class` to `className`! A `breadcrumbs` prop can be passed to `<Layout>` to set the dynamic page title. Passing props is similar to an attribute on an HTML element:

```
import React from "react"
import Layout from "../components/Layout"
import { Link } from "gatsby"

export default function EllaArborist() {
  return (
    {/* highlight-start */}
    <Layout breadcrumbs={["Who We Are", "Ella"]}>
    {/* highlight-end */}
      <h1>Ella - Arborist</h1>
      <h2>Ella is an excellent Arborist. We guarantee it.</h2>
      <div className="bio-card">
        <img
          alt="Comically crude stick person sketch"
          src="/person.png"
        />
        <p>Ella</p>
      </div>
    </Layout>
```

```
  );
}
```

The other 2 `Who We Are` pages for Marin and Sam can now be made with a similar structure. Maybe you are even thinking about another component for the Bio Card!

Once `/services` and the root level HTML files are ported, here is what the finished Gatsby project file structure looks like:

```
gatsby-site
├── static
│   ├── favicon.ico
│   └── person.png
├── src
│   ├── styles
│   │   ├── normalize.css
│   │   └── style.css
│   ├── components
│   │   └── Layout.js
│   └── pages
│       ├── index.js
│       ├── about.js
│       ├── contact.js
│       ├── 404.js
│       ├── who
│       │   ├── index.js
│       │   ├── ellla-arborist.js
│       │   ├── marin-leafer.js
│       │   └── sam-surgeon.js
│       └── services
│           ├── index.js
│           ├── growing.js
│           ├── cleaning.js
│           └── shrinking.js
├── node_modules
├── .gitignore
├── .prettierignore
├── .prettierrc
├── gatsby-config.js
├── LICENSE
├── package-lock.json
├── package.json
└── README.md
```

## Building & Deploying

With your new Gatsby application taking shape, it's time to integrate it into your existing HTML website.

### Gatsby build step

With all of the pages ported over, you now have a site that mirrors the existing HTML site. Stop the development server if it's still running; it's time to run the production build! 🚀

```
gatsby build
```

Once a build is complete, the compiled set of files can be found in `/public`.

### Hosting the new website

Once built, the contents of the `/public` folder are ready to be hosted at the root ( `/` ) of a domain. The files can be deployed in similar ways to how your existing HTML site may have been deployed. For more deployment options including automation with git and cloud services, see the Deploying and Hosting page.

What about migrating a section of a site? No problem. The Gatsby build output in `/public` can be mixed with existing files.

If the Gatsby site is to be hosted at a non-root path, e.g. `example.com/blog/` , Gatsby needs to be informed so page and asset links in the built output can be prefixed.

Path prefix is the Gatsby option for setting a non-root hosting path. Here is the example project `gatsby-config.js` file with a path prefix added for `/blog` :

```
/**
 * Configure your Gatsby site with this file.
 *
 * See: https://www.gatsbyjs.com/docs/reference/config-files/gatsby-config/
 */

module.exports = {
  pathPrefix: "/blog", // highlight-line
  plugins: ["gatsby-plugin-react-helmet"],
}
```

If the HTML and non-HTML files of the Gatsby build output are to be served in different locations, this can also be supported with the Asset Prefix option.

Once prefix options have been set, the build must be run with the `--prefix-paths` flag to apply them:

```
gatsby build --prefix-paths
```

You can also serve the built site locally for review using the `--prefix-paths` flag:

```
gatsby serve --prefix-paths
```

### New website file structure

Here is the structure of the HTML & non-JavaScript asset files in the Gatsby build output:

```
website-domain
  ├── favicon.ico
```

```
├── person.png
├── index.html
├── 404
│   └── index.html
├── about
│   └── index.html
├── contact
│   └── index.html
├── services
│   ├── index.html
│   ├── growing
│   │   └── index.html
│   ├── cleaning
│   │   └── index.html
│   ├── shrinking
│   │   └── index.html
└── who
    ├── index.html
    ├── ellla-arborist
    │   └── index.html
    ├── marin-leafer
    │   └── index.html
    └── sam-surgeon
        └── index.html
```

## Next steps

Gatsby can handle images through direct imports to page and component files too! The asset import documentation covers this. There is also gatsby-plugin-image component for even deeper optimizations. Once assets are handled through Gatsby, plugins can be used to optimize their processing and delivery.

The building with components doc has information about why Gatsby uses React component architecture and how it fits into a Gatsby application.

Sourcing content and data is a great next step if you are interested in separating your content from your website code, such as sourcing the site title from `gatsby-config.js` with GraphQL and writing content in Markdown.

Short guides can be found at the recipes section for adding functionality such as optimizing and querying local images, adding Markdown support and integrating various modern CSS tools. The adding website functionality page has longer guides for larger tasks such as making your site accessible, adding authentication, and fetching data with client-side JavaScript.

Gatsby is dedicated to making you feel welcome! Learn more and engage with the community by starting a conversation or contributing yourself. The contributing page has further information and channels where you can get support.