Deploying a React Project (Vite) to GitHub + Git Branching Guide

Amr A Khllaf

August 1, 2025

Contents

1 Deploying a React Project (Vite) to GitHub + Git Branching Guide

This guide explains how to:

1. Set up a React project using Vite.
2. Deploy the project to GitHub Pages.
3. Understand the difference between HashRouter and BrowserRouter.
4. Use Git and GitHub effectively, including branches and commits.

1.1 1. Setting up a Vite React Project

Vite is a fast build tool for modern web projects. It helps you scaffold and develop React apps faster than Create React App (CRA).

1.1.1 Steps:

```
npm create vite@latest
```

- Choose **React** and **JavaScript** when prompted.

Then:

```
cd your-project-name
npm install
```

Vite provides instant reload, faster dev server, and smaller build size.

1.2 2. Preparing for GitHub Pages Deployment

To make the project work when hosted on GitHub Pages, some configuration is needed.

1.2.1 Step 1: Update `vite.config.js`

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  base: "/your-repo-name/", // Replace with your GitHub repo name
});
```

Why this is needed: GitHub Pages serves the site from a subfolder (`https://username.github.io/repo-`
Setting the `base` makes Vite use correct asset paths.

1.2.2 Step 2: Add `homepage` field to `package.json`

This ensures the correct base path is used when building the app:

```
{
  "homepage": "http://your-username.github.io/your-repo-name",
  "name": "ecommerce",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "predeploy": "npm run build",
    "deploy": "gh-pages -d dist"
  }
}
```

Why this is needed: The `homepage` field helps some tools like `gh-pages` and build systems know where the site will be hosted. It avoids broken paths when loading static files.

1.3 3. Choosing the Right Router: HashRouter vs. BrowserRouter

In React Router (v6), there are two main types of routers:

1.3.1 BrowserRouter

- Uses the HTML5 history API
- Clean URLs like `/login`
- Requires server-side support (e.g. Node, Express)

1.3.2 HashRouter

- Uses hash-based URLs like `/#/login`
- Works on static hosts like GitHub Pages
- No server configuration needed

For GitHub Pages, always use `HashRouter`.

1.3.3 Example:

```
import { HashRouter } from "react-router-dom";

<HashRouter>
  <App />
</HashRouter>;
```

1.3.4 Problem Solved by HashRouter:

GitHub Pages doesn't handle dynamic routing like a backend server. If you use `BrowserRouter`, refreshing or accessing a nested route (like `/about`) will result in a 404 error.

`HashRouter` avoids this by keeping everything after the `#` in the URL, so GitHub Pages serves `index.html` correctly no matter what route you're on.

1.4 4. Install Deployment Tool

Install the `gh-pages` package to handle deployment:

```
npm install --save-dev gh-pages
```

This tool pushes the production-ready code inside `dist/` to a special `gh-pages` branch.

1.5 5. First Push to GitHub

1.5.1 Steps:

1. Create a repository on GitHub (e.g., `fresh-cart-ecommerce`).
2. In your terminal:

```
git init
git remote add origin https://github.com/your-username/your-repo-name.git
git add .
git commit -m "initial commit"
git push -u origin main
```

Why this is needed: This links your local project with the remote GitHub repository and pushes the initial code.

1.6 6. Deploy to GitHub Pages

Once everything is set up, run the deployment command:

```
npm run deploy
```

This will:

1. Build your project using `npm run build`, generating optimized production code in the `dist/` folder.
2. Automatically push the contents of `dist/` to the `gh-pages` branch using `gh-pages`.

1.6.1 After Deployment:

- Go to GitHub repository → Settings → Pages
- Choose `gh-pages` branch as the source
- Click "Save"

After a few minutes, your app should be live at:

`https://your-username.github.io/your-repo-name`

1.7 7. Making Changes and Committing

1.7.1 A. Pushing to `main` branch:

```
git add .
git commit -m "Describe what was updated or fixed"
git push origin main
```

1.7.2 B. Working on a separate feature branch:

```
git checkout -b feature/login
```

This creates a new branch for login functionality.

```
git add .
git commit -m "Finish login flow"
git push origin feature/login
```

Why use branches?

- Safer development without breaking `main`
- Easy to manage features or bug fixes
- Encouraged in collaborative teams

To merge back into `main`, create a Pull Request on GitHub or use:

```
git checkout main
git merge feature/login
```

1.8 8. Summary of Useful Git Commands

Purpose	Command
Initialize Git repo	`git init`
Add remote repo	`git remote add origin <url>`
Add all changes	`git add .`
Commit with message	`git commit -m "message"`
Push to main	`git push origin main`
Create new branch	`git checkout -b feature/branch-name`
Push branch	`git push origin feature/branch-name`

1.9 Conclusion

This guide helps you:

- Set up a modern Vite + React project
- Use the correct router for static hosting
- Deploy with GitHub Pages
- Work efficiently with Git branches and commits

Following these steps ensures your app is production-ready and easily accessible online.