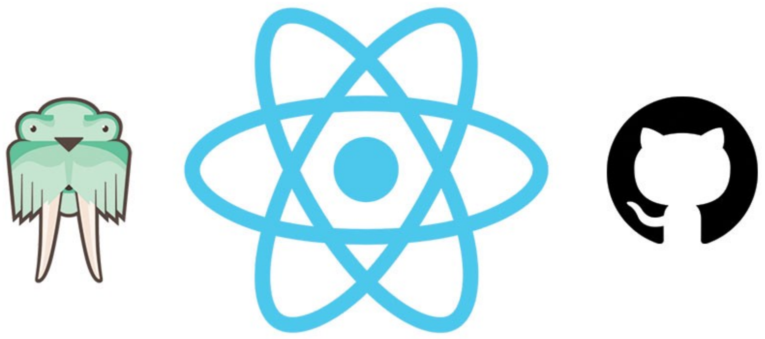
Jake Wiesler Follow
front-end developer • human cyborg relations
Dec 11, 2016 · 8 min read

1.
Surge VS GitHub Pages: How to deploy a create-react-app project



As a developer, there are several ways you can show off your skills to peers and prospective employers. Open source contributions are great. Blogging is great. But at some point you'll want to get projects up on the web, where people can actually use them.

Too many times have I started something locally and never seen it through. You may have done the same. One reason this is so common is because of all the extra work that goes into deployment.

Wouldn't it be nice to have a central hub where all of your projects live without worrying about hosting and server configuration? This post will walk you through two popular deployment tools that require a minimum amount of effort.

Be the turtle.

create-react-app

Before you get started, let me tell you about my new best friend, `create-react-app`. If you've been thinking of learning React—but are taken back by the amount of time it requires to get a project up and running—this command line interface (CLI) tool is your saving grace. It will take care of most of the boilerplate you need to start a project.

To install it and create your first project, go into your terminal and run these commands:

```
npm install -g create-react-app

create-react-app <your project name>

cd <your project name>
```

Out of the box `create-react-app` comes with a few handy scripts that allow you to develop your project locally and deploy it afterwards. They can be found in the `package.json` file at the root of your project directory.

Use `npm start` to run your project locally while you develop it. Then use `npm run build` to prepare your project for deployment.

Surge.sh and GitHub Pages

Let's fast forward a bit. You've built a basic app and are ready to deploy it to the web.

There are many options in the realm of static site hosting platforms, but the two we'll be working with are Surge.sh and GitHub Pages.

Both of these platforms are powerful in their own right. Which one you use comes down to your situation. My goal is to give you a better understanding as to why these tools exist, and what you can do with them.

Also note that even though this post is all about publishing projects created with the `create-react-app` CLI, Surge and GitHub Pages will work with even the most basic of projects. You may be able to skip some of these steps if you aren't using React itself.

Surge.sh

Surge is a really awesome piece of software that I recently discovered from a thread on Reddit. At its core, Surge is a CLI that allows you to deploy your projects for free. And quickly, too. What makes Surge really stand out is its simplicity.

Let's walk through a simple example using `create-react-app` .

First, install the `surge` package globally:

```
npm install -g surge
```

Now that Surge has been installed on your machine, you need to prepare your project for deployment. I mentioned above that `create-react-app` has a script in `package.json` called `build` . This script essentially prepares the application for production by bundling and optimizing all of the code.

Run `npm run build` in your project's root:

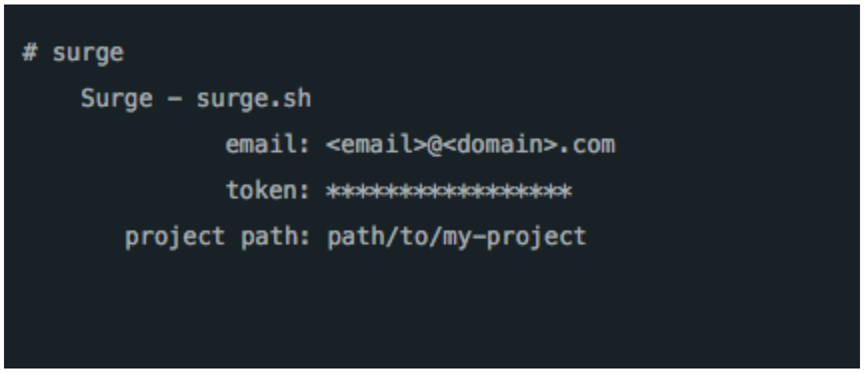
```
npm run build
```

You should notice that a new folder has been created in the root of your project directory called `build` . This folder contains the production-ready application.

Excellent, you're almost finished. All that's left is to run the `surge` command in your project's root:

```
surge
```

If this is your first time running `surge` you'll be prompted to create an account. Add an email and password, then hit enter. You'll then see output similar this:



```
# surge
    Surge - surge.sh
              email: <email>@<domain>.com
              token: *****************
       project path: path/to/my-project
```

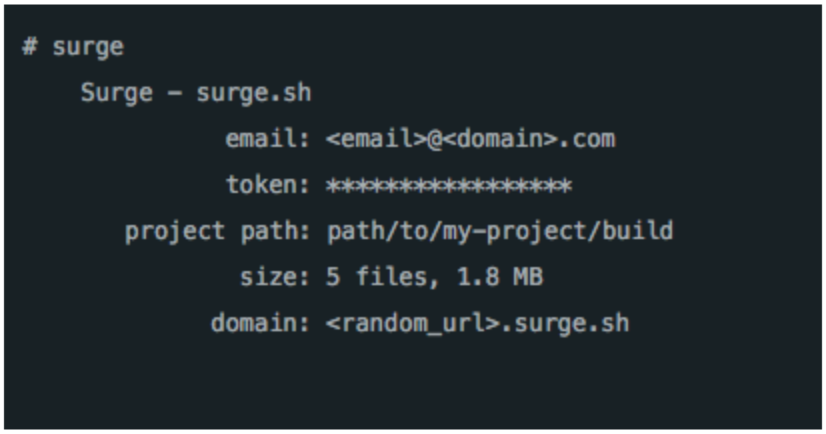
To deploy your project, Surge only needs two things from you:

- The path to the project
- The domain to host it on

Project path

You'll notice that the *project path* field in the terminal defaults to the root directory. Surge assumes that the whichever directory you're running the `surge` command in is the directory you wish to deploy. In your case, you need to point Surge to the `build` directory created when you ran `npm run build` .

If your project path is `path/to/my-project` , edit this to `path/to/my-project/build` . Once you've made this change, press enter to confirm.

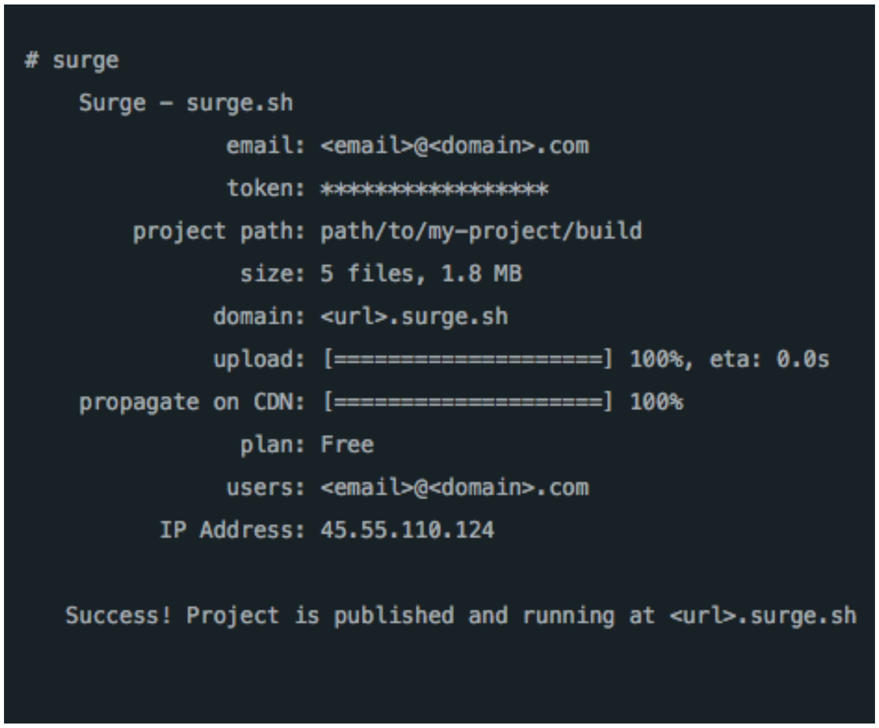


```
# surge
    Surge - surge.sh
              email: <email>@<domain>.com
              token: *****************
       project path: path/to/my-project/build
               size: 5 files, 1.8 MB
             domain: <random_url>.surge.sh
```

Domain

After entering the project path, Surge will suggest a random domain to use. You can delete it and add your own domain if you wish. It just needs to have the `.surge.sh` extension at the end. The tool also allows for custom domains, which is really awesome.

Accept the suggested domain, or add your own (custom or random with correct surge extension), then hit enter.



```
# surge
    Surge - surge.sh
               email: <email>@<domain>.com
               token: *****************
        project path: path/to/my-project/build
                size: 5 files, 1.8 MB
              domain: <url>.surge.sh
              upload: [====================] 100%, eta: 0.0s
    propagate on CDN: [====================] 100%
                plan: Free
               users: <email>@<domain>.com
          IP Address: 45.55.110.124

   Success! Project is published and running at <url>.surge.sh
```

That's all she wrote! Navigate to the domain in your browser and you should see your project up and running.

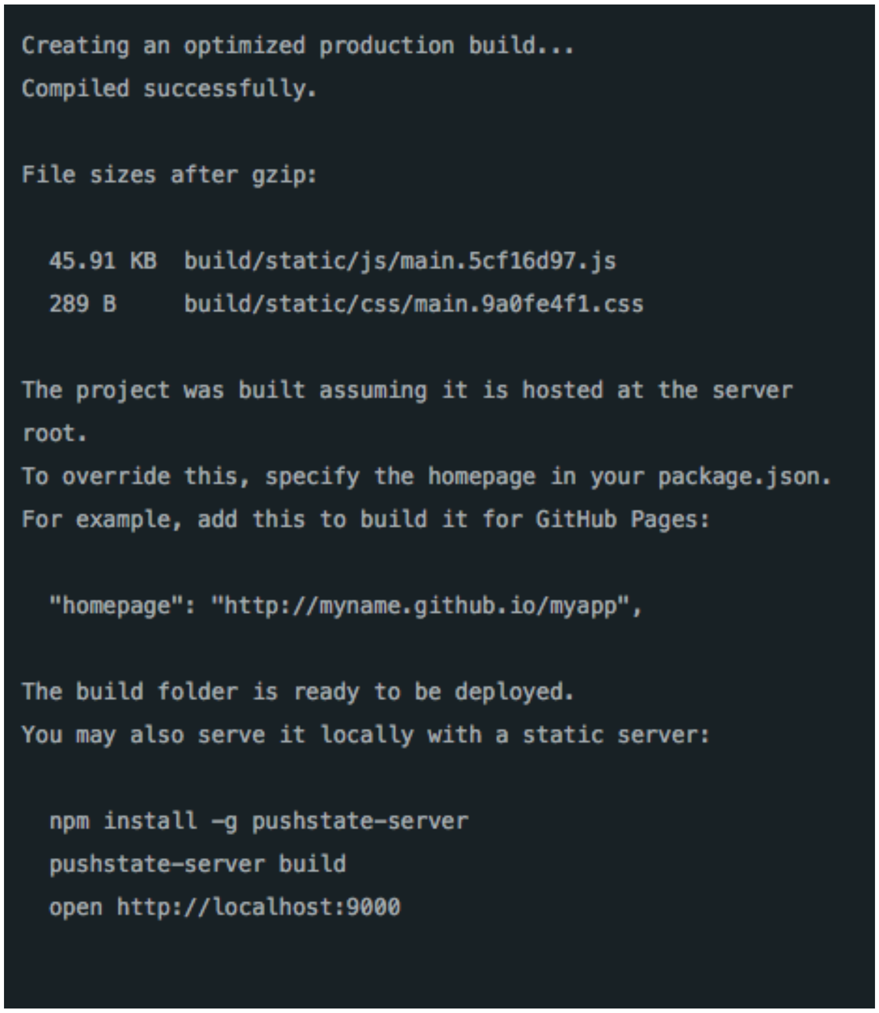
Note that if your application is using **client-side-routing**, Surge recommends you rename the `index.html` file in your `build` directory to `200.html` before running the `surge` command. You can find out more information in the Surge documentation.

GitHub Pages

GitHub Pages makes it easy to turn GitHub repositories into full fledged static websites. Many organizations use this service to host their documentation and project demos, but you can use it for whatever you'd like.

Note that in order for this to work, your must first push the code to a repository on GitHub. If this sounds foreign to you, check out further documentation here.

If you've ever run `npm run build` using `create-react-app` before, then you may have noticed output that looks like this:



```
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  45.91 KB  build/static/js/main.5cf16d97.js
  289 B     build/static/css/main.9a0fe4f1.css

The project was built assuming it is hosted at the server
root.
To override this, specify the homepage in your package.json.
For example, add this to build it for GitHub Pages:

  "homepage": "http://myname.github.io/myapp",

The build folder is ready to be deployed.
You may also serve it locally with a static server:

  npm install -g pushstate-server
  pushstate-server build
  open http://localhost:9000
```

initial output after running`npm run build`

`create-react-app` comes with detailed documentation to help users publish their work using all sorts of tools. Here you can see real time terminal output instructing us on how to do so via GitHub Pages. Let's try it out.

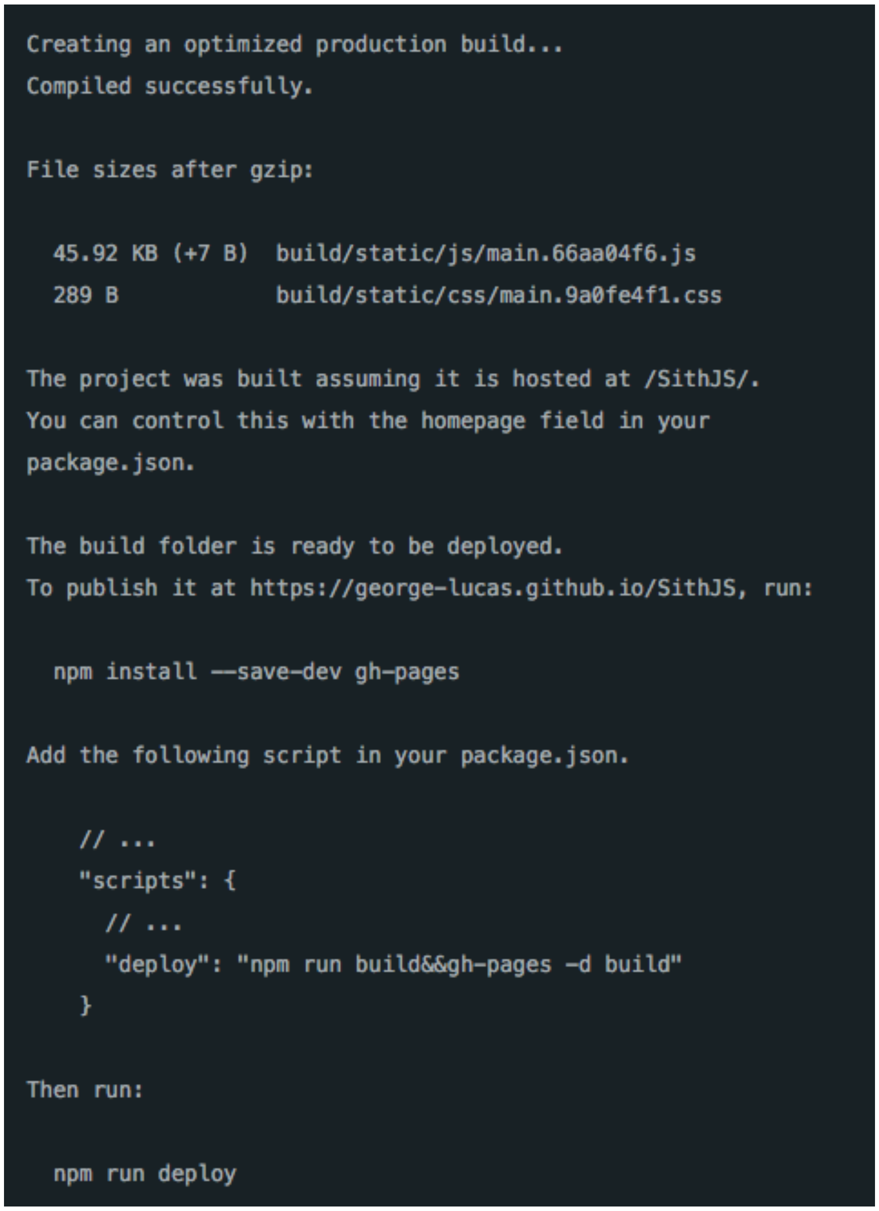
Step 1

Edit `package.json` by adding a new field named `homepage`:

```
"homepage": "https://<github-username>.github.io/<project-
repo>"
```

If your GitHub username is `george-lucas`, and your project's GitHub repository is `SithJS`, the value of the `homepage` field should be `"https://george-lucas.github.io/SithJS"`.

Let's run `npm run build` again after the change:



```
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  45.92 KB (+7 B)  build/static/js/main.66aa04f6.js
  289 B            build/static/css/main.9a0fe4f1.css

The project was built assuming it is hosted at /SithJS/.
You can control this with the homepage field in your
package.json.

The build folder is ready to be deployed.
To publish it at https://george-lucas.github.io/SithJS, run:

  npm install --save-dev gh-pages

Add the following script in your package.json.

    // ...
    "scripts": {
      // ...
      "deploy": "npm run build&&gh-pages -d build"
    }

Then run:

  npm run deploy
```

new output after adding a `homepage` field in `package.json`

Did you notice the new output above? The `create-react-app` CLI is walking us through the entire process. Pretty snazzy.

Step 2

Next you need to install the `gh-pages` plugin. This will allow us to publish to the `gh-pages` branch on GitHub straight from within the terminal:

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

`gh-pages` is a special branch that GitHub Pages uses to publish projects. The beautiful thing about it is that the branch lives in the same repository as your project's code, but doesn't affect the project itself.

Note that if you already have a `gh-pages` branch in your project's repository, it will update the branch accordingly. If the branch doesn't exist, it will create it on the fly.

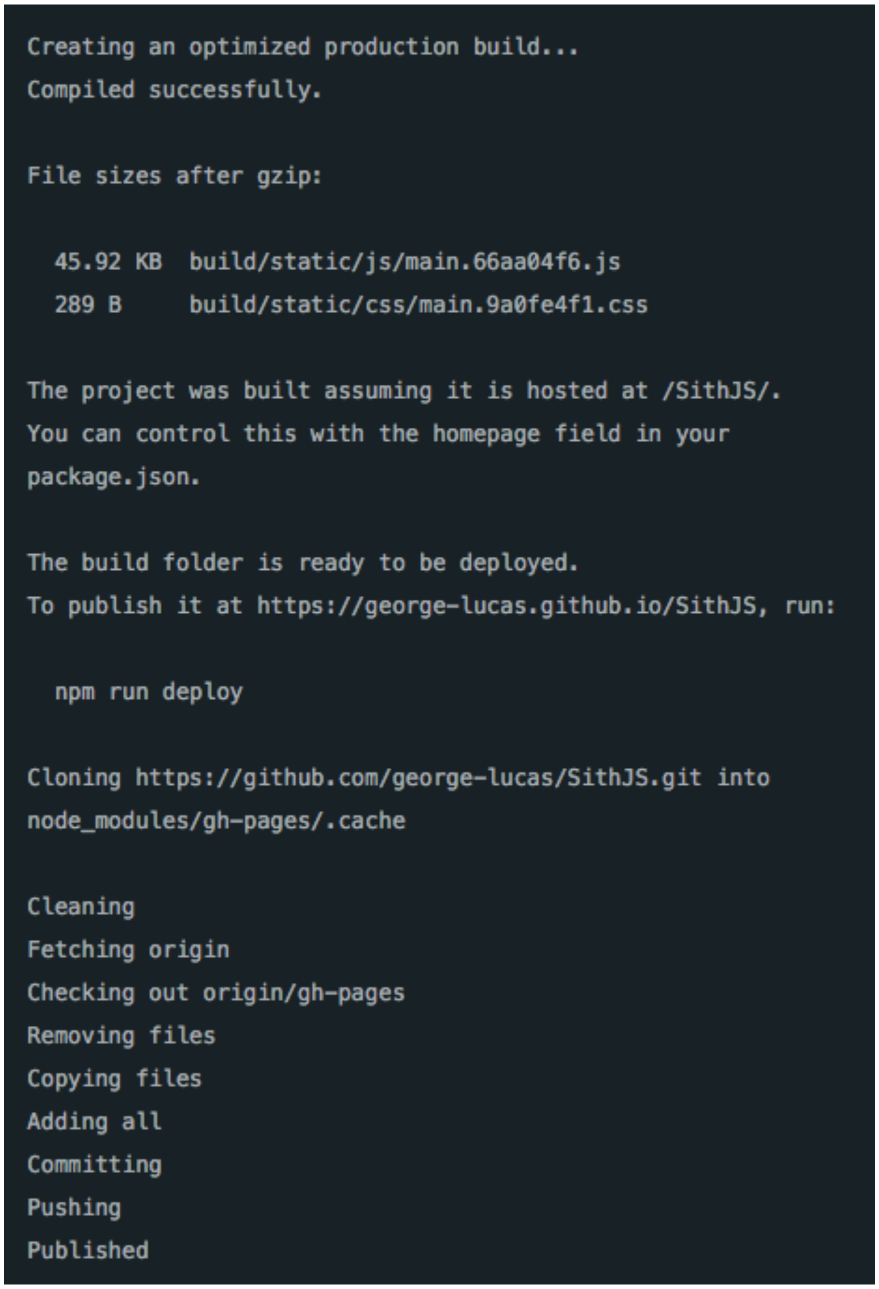
Step 3

Add a new script to the `scripts` field inside `package.json`. Let's call the script `deploy`:

```
"deploy" : "npm run build&&gh-pages -d build"
```

And finally let's run it:

```
npm run deploy
```

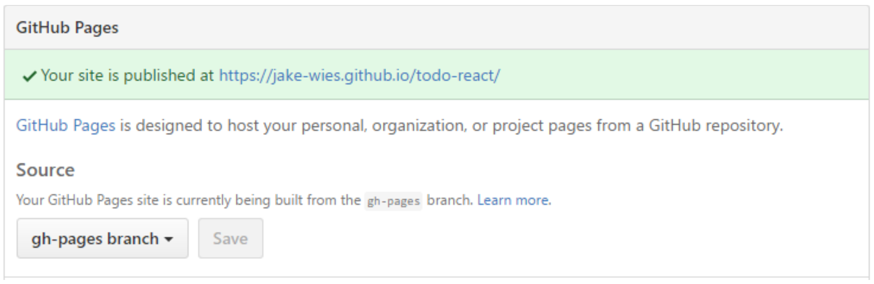


```
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  45.92 KB  build/static/js/main.66aa04f6.js
  289 B     build/static/css/main.9a0fe4f1.css

The project was built assuming it is hosted at /SithJS/.
You can control this with the homepage field in your
package.json.

The build folder is ready to be deployed.
To publish it at https://george-lucas.github.io/SithJS, run:

  npm run deploy

Cloning https://github.com/george-lucas/SithJS.git into
node_modules/gh-pages/.cache

Cleaning
Fetching origin
Checking out origin/gh-pages
Removing files
Copying files
Adding all
Committing
Pushing
Published
```

`npm run deploy` will first build your project via `npm run build`. Then it will publish it to a `gh-pages` branch on GitHub via `gh-pages -d build`.

Step 4

We're almost done. Head to your project's repository settings on GitHub. In the **GitHub Pages** section, confirm that your project is set to use the `gh-pages` branch.


GitHub Pages

✓ Your site is published at https://jake-wies.github.io/todo-react/

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Source

Your GitHub Pages site is currently being built from the `gh-pages` branch. Learn more.

An example of my React todo list using the gh-pages branch

You can now navigate to the URL you entered in the `homepage` field of your `package.json` file, where you'll see your project has been deployed!

Note that—like with Surge—GitHub Pages also has trouble with client-side routing. `create-react-app` lists a few solutions in the documentation for GitHub Pages integration.

The winner

In all honesty, you can't go wrong with either of these options. They're both great. Let's recap some key features of each:

Surge

Minimal configuration to deploy a project

Makes no assumptions about technology used

Seamless integration with build tools such as Grunt and Gulp

Can be used as a development dependency when building your own tools

GitHub Pages

Keeps project code and webpage(s) housed in a single repository

Centralizes all projects underneath your *<username>.github.io* domain

Deploy from the command line or from your repository settings on GitHub

Works great with static site generators like Jekyll

Personally, I chose GitHub Pages for my most recent project because I already use GitHub on a daily basis and I like to keep everything centralized. Maybe it's my OCD talking, but I love having individual GitHub repositories for projects that I can deploy as a subdomain of `jake-wies.github.io` .

If you're simply creating a test project, or you want to show a demo to a client, using Surge's super-fast CLI to generate a webpage is hard to pass up. You can generate the domain quickly and tear it down afterwards.

At the end of the day, the best tool for the job is the one that makes you productive. The information I've provided should give you a good understanding of where each shines. Audit your project needs and choose which is right for you.

Thanks for reading! I'm a self-taught developer & spend most of my free time diving into front-end tools and writing my way out.