Placement Empowerment Program Cloud Computing and DevOps Centre

Deploy Your Static Website Using GitHub Pages
Host your local Git repository's static website directly using GitHub Pages.

Name:Balasubramanian

St. JOSEPH'S COLLEGE OF ENGINEERING | St. JOSEPH'S INSTITUTE OF TECHNOLOGY

AUTONOMOUS INSTITUTIONS, AFFILIATED TO ANNA UNIVERSITY

GUIDE TO GITHUB PAGES SETUP

INTRODUCTION TO GITHUB PAGES

GitHub Pages is a static website hosting service offered by GitHub, allowing users to create and publish websites directly from their GitHub repositories. This service is particularly beneficial for web developers, businesses, and individuals looking to establish an online presence without the complexities associated with traditional web hosting solutions. By leveraging GitHub's robust version control and collaboration features, users can easily manage their website content and collaborate with others.

One of the primary purposes of GitHub Pages is to host project documentation, portfolios, blogs, or any static content. Unlike dynamic web hosting services that require server-side processing, GitHub Pages serves static files directly from a repository, enabling faster loading times and improved performance. This makes it an ideal solution for developers who want to showcase their work or share information about their projects without the overhead of managing server infrastructure.

The benefits of using GitHub Pages extend beyond simple hosting. For web developers, it integrates seamlessly with Git repositories, allowing for smooth version control and easy updates. Users can push changes to their repository, triggering automatic updates to their live site. This workflow not only simplifies the process of deploying updates but also enhances collaboration, as multiple contributors can work on the same project without conflicts. Moreover, GitHub Pages supports custom domains, enabling businesses to create branded experiences without sacrificing the ease of use and functionality provided by the platform. With built-in support for Jekyll, a static site generator, users can also take advantage of templating and content management features, streamlining the development of complex sites.

Overall, GitHub Pages stands out as a powerful tool for anyone looking to create a professional online presence with minimal effort and maximum control.

SETTING UP YOUR LOCAL REPOSITORY

Creating a local Git repository is the first step in managing your project with version control. Follow these steps to initialize a Git repository, add files, commit changes, and link it to a remote repository on GitHub.

STEP 1: INSTALL GIT

Before you start, ensure that Git is installed on your machine. You can download it from the official Git website and follow the installation instructions for your operating system.

STEP 2: INITIALIZE A GIT REPOSITORY

Navigate to your project directory using the command line or terminal. Once inside the desired folder, run the following command to initialize a new Git repository:

```
git init
```

This command creates a new subdirectory named `.git`, which contains all the necessary files for version control.

STEP 3: ADD FILES TO THE REPOSITORY

After initializing your repository, you can start adding files. To stage a specific file, use the command:

```
git add filename
```

To stage all files in the directory, use:

```
git add .
```

This prepares the files for committing by adding them to the staging area.

STEP 4: COMMIT CHANGES

Once your files are staged, you need to commit them to the repository. This step captures a snapshot of your current project state. Use the following command:

```
git commit -m "Your commit message here"
```

Make sure to write a clear and concise commit message that describes the changes made.

STEP 5: LINK TO A REMOTE REPOSITORY ON GITHUB

To push your local repository to GitHub, you first need to create a remote repository on GitHub. After creating the repository, link your local repository to it using:

```
git remote add origin https://github.com/username/repository.git
```

Replace `username` and `repository` with your GitHub username and the name of your repository. Finally, push your local commits to GitHub with:

```
git push -u origin master
```

This command uploads your commits to the remote repository, establishing a connection between your local and remote repositories. With these steps, you are now set up to manage your project using Git effectively.

CREATING A STATIC WEBSITE

Designing a static website involves using fundamental web technologies such as HTML, CSS, and JavaScript. A static website consists of fixed content, which means that the same HTML code is delivered to every visitor. This simplicity allows for quick loading times and straightforward deployment, particularly on platforms like GitHub Pages.

BASIC HTML STRUCTURE

The backbone of your static website is HTML (HyperText Markup Language). Begin by creating an index.html file, which serves as the homepage of your site. Here's a simple structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Your Website Title</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Welcome to My Website</h1>
        <nav>
            <ul>
                <li><a href="#about">About</a></li>
                <li><a href="#services">Services</a></li>
                <li><a href="#contact">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <section id="about">
            <h2>About Us</h2>
            <p>Information about your website.</p>
        </section>
        <section id="services">
            <h2>Our Services</h2>
            <p>Details on services offered.</p>
        </section>
        <section id="contact">
            <h2>Contact Us</h2>
            <p>Contact information.</p>
        </section>
    </main>
    <footer>
```

```
            <p>&copy; 2023 Your Name. All rights reserved.</
p>
        </footer>
        <script src="script.js"></script>
</body>
</html>
```

STYLING WITH CSS

Next, create a styles.css file to add style to your website. CSS (Cascading Style Sheets) controls the layout and appearance of your HTML elements. Here's a basic example:

```
body {
        font-family: Arial, sans-serif;
        line-height: 1.6;
        margin: 0;
        padding: 0;
}   header
{

        background: #333;
        color: #fff;
        padding: 10px 0;
        text-align: center;
} nav ul {


        list-style: none;
        padding: 0;
} nav ul li {


        display: inline;
        margin: 0 10px;
}  main
{
```

```
        padding: 20px;
}
```

ADDING INTERACTIVITY WITH JAVASCRIPT

For added interactivity, create a script.js file. JavaScript allows you to manipulate the HTML elements dynamically. Here's a simple example that displays an alert when the user clicks a button:

```
document.addEventListener("DOMContentLoaded", function() {

        const button = document.createElement("button");
        button.textContent = "Click Me!";
        button.onclick = function() {
                alert("Hello, welcome to my website!");
});      };
        document.body.appendChild(button);
```

FILE STRUCTURE AND BEST PRACTICES

Organizing your project files is essential for maintainability. A recommended structure is:

```
/your-project  |--
index.html     |--
styles.css     |--
script.js      |--
/images
```

1. **Keep your files organized**: Separate HTML, CSS, and JavaScript files to maintain clarity.
2. **Use meaningful file names**: This makes it easier to identify the purpose of each file.
3. **Comment your code**: Adding comments helps others (and your future self) understand your thought process.

By following these guidelines, you'll create a well-structured and visually appealing static website ready for deployment on GitHub Pages.

PUBLISHING YOUR WEBSITE WITH GITHUB PAGES

Configuring GitHub Pages for your repository is a straightforward process that allows you to publish your static website effortlessly. Here's a step-by-step guide to get your site live.

STEP 1: ACCESS REPOSITORY SETTINGS

Begin by navigating to your GitHub repository where your project files are stored. Click on the **"Settings"** tab located at the top of the repository page. This section contains various options to manage your repository settings.

STEP 2: FIND THE GITHUB PAGES SECTION

Scroll down to the **"Pages"** section within the settings. This is where you'll configure how your site will be published. You will see an option to select the source of your GitHub Pages.

STEP 3: SELECT A BRANCH

In the GitHub Pages section, you can choose the branch that will be used to publish your site. Typically, this is the main or master branch, but you can also create a separate branch, such as gh-pages, specifically for your website files. Select the branch from the dropdown menu and, if applicable, choose the / (root) folder or /docs folder where your HTML files are stored.

STEP 4: CUSTOM DOMAIN SETUP (OPTIONAL)

If you want to use a custom domain instead of the default username.github.io/repository, you can set this up in the same section. Click on the **"Custom domain"** field and enter your domain name. After entering the domain, remember to configure your DNS settings to point to GitHub's servers by adding the necessary A and CNAME records provided by GitHub.

STEP 5: SAVE CHANGES

After making your selections, click the **"Save"** button to apply your changes. GitHub will then generate a URL for your published site, typically in the format https://username.github.io/repository.

STEP 6: VIEW YOUR LIVE SITE

Once you have saved your settings, the page will refresh, and you'll see a message indicating that your site is published. You can view your live site by clicking the provided link. It may take a few minutes for your changes to appear, especially if you just set up a custom domain.

By following these steps, you can efficiently configure GitHub Pages to host your static website, making it accessible to the world.

MAINTAINING AND UPDATING YOUR SITE

Once your static website is live on GitHub Pages, regular maintenance and updates are essential to keep it relevant and functional. Here are some effective strategies to ensure your site remains up-to-date.

MAKING CHANGES LOCALLY

To maintain your site, you should always make changes in your local development environment before pushing updates to GitHub. This allows for testing and troubleshooting without affecting the live site. You can edit your HTML, CSS, and JavaScript files in your preferred code editor. After making changes, preview your updates in a browser to ensure everything looks and functions as intended.

PUSHING UPDATES TO GITHUB

Once you've confirmed your changes are working locally, it's time to push these updates to your GitHub repository. First, open your terminal or command prompt, navigate to your project directory, and follow these steps:

1. Stage your changes by using `git add .` to add all modified files.
2. Commit the changes with a descriptive message: `git commit -m "Updated content on homepage"`.
3. Push the changes to the remote repository using `git push origin main` (or replace `main` with the branch you are working on).

MANAGING VERSION CONTROL

Version control is a critical aspect of maintaining your website. Git allows you to track changes over time, making it easy to revert to previous versions if something goes wrong. Use `git log` to view your commit history, and `git`

checkout to switch between different commits or branches. Regular commits with meaningful messages will help you maintain clarity about the evolution of your project.

LEVERAGING BRANCHES FOR DEVELOPMENT

Using branches is an effective way to manage new features or updates without disrupting the main version of your site. Create a new branch for each feature or update using the command git checkout -b feature- branch-name. This way, you can work on changes independently and merge them back into the main branch once they are complete and tested.

When updates are ready to be incorporated into the main site, use git checkout main to switch back to the main branch, followed by git merge feature-branch-name to integrate your changes. This workflow minimizes conflicts and allows for organized development.

By following these strategies, you can ensure that your website remains current and continues to meet the needs of your visitors.