GitHub for Beginners

Comprehensive Guide to Collaboration

Scott Tremaine
Software Developer and Educator

Breakpoint Coding Tutorials
© 2024 by John Scott Tremaine. All rights reserved.

Contents

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Overview and Purpose of GitHub

GitHub is a web-based platform that leverages the power of Git, a distributed version control system, to facilitate collaborative software development. It provides a user-friendly interface and a suite of tools designed to enhance the software development workflow. GitHub allows developers to host their code repositories, manage version control, collaborate with other developers, and deploy projects, all in one place.

Primary Purpose of GitHub

The primary purpose of GitHub is to streamline the development process by providing tools that enable developers to work together more efficiently. By hosting code on GitHub, teams can keep track of changes, manage project versions, and collaborate seamlessly, regardless of geographical location.

Key Aspects of GitHub

- GitHub hosts millions of repositories, ranging from small personal projects to large-scale enterprise software.
- GitHub facilitates collaboration through features like pull requests, code reviews, and issue tracking.
- GitHub provides tools for managing tasks, tracking progress, and organizing work using issues and project boards.
- With GitHub Actions, developers can automate testing, building, and deploying their applications.

Differences between Git and GitHub

While Git and GitHub are closely related, they serve different purposes in the development workflow.

Git

Git is a distributed version control system (DVCS) developed by Linus Torvalds in 2005. It is a command-line tool that allows developers to track changes in their source code over time. Git enables developers to create branches, merge changes, and manage versions of their codebase. It operates locally on a developer's machine, allowing for offline work and faster operations.

GitHub

GitHub is a web-based platform built on top of Git that provides a graphical interface and additional features for managing Git repositories. It facilitates collaboration by providing tools for code review, issue tracking, and project management. GitHub hosts repositories in the cloud, making it easy to share code and collaborate with others. It integrates with various tools and services to enhance the development workflow, including continuous integration (CI) and continuous deployment (CD) pipelines.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Relation between Git and GitHub

Git is the underlying version control system, while GitHub is a platform that makes it easier to use Git for collaborative development.

Key Features of GitHub

GitHub offers a variety of features that make it a powerful tool for developers.

Repositories

A repository (or repo) is a storage space where your project's files and their revision history are kept.

Developers can create repositories to host their code. Repositories can be public (accessible to anyone) or private (restricted access).

GitHub provides settings to manage repository visibility, collaborators, and integrations.

Branches

Branches are a core feature of Git that allow you to diverge from the main line of development and continue to work without affecting that main line.

Developers often use branches to develop features, fix bugs, or experiment with new ideas. Once the work on a branch is complete, it can be merged back into the main branch.

Pull Requests

A pull request is a method of submitting contributions to a repository. It lets you notify project maintainers about changes you've pushed to a branch in a GitHub repository.

Pull requests facilitate code review and discussion before merging changes into the main branch. They provide a way to propose changes, review the differences, and integrate them after approval.

Issues

Issues are used to track tasks, enhancements, and bugs for your projects.

Issues can be assigned to team members, labeled for categorization, and linked to pull requests and commits. They are a key tool for project management and communication within a development team.

Project Boards

Project boards are customizable boards that can be used to organize and prioritize your work.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Similar to Kanban boards, project boards allow developers to create and manage tasks, track their progress, and ensure that work is aligned with project goals.

GitHub Actions

GitHub Actions is a CI/CD platform that allows you to automate your build, test, and deployment pipeline.

Developers can create workflows that automatically build and test code, deploy applications, and handle other repetitive tasks.

GitHub Pages

GitHub Pages is a static site hosting service that takes HTML, CSS, and JavaScript files directly from a repository on GitHub, optionally runs the files through a build process, and publishes a website.

It's commonly used for project documentation, personal websites, and portfolio projects.

Wikis

Every GitHub repository comes with a wiki that you can use to share long-form content about your project.

Wikis are useful for documentation, how-tos, and other detailed information related to the project.

By leveraging these features, developers can effectively manage their code, collaborate with team members, and streamline their development workflow.

Creating a GitHub Account

Step-by-step Guide to Creating a GitHub Account

Visit the GitHub Website

Open your web browser and go to GitHub's website.

Sign Up for an Account

1. Click on the "Sign up" button located in the upper-right corner of the homepage.
2. Enter your desired username, email address, and a strong password.
3. Click on "Create account" to proceed.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Verify Your Email

GitHub will send a verification email to the address you provided. Check your email inbox and follow the instructions in the email to verify your account.

Complete the Setup

After verifying your email, you will be prompted to complete a few more steps to set up your account. This includes selecting your plan (free or paid) and personalizing your experience. Follow the on-screen instructions to complete these steps.

Welcome to GitHub

Once your account is set up, you will be directed to your GitHub dashboard, where you can start creating and managing repositories.

Configuring Your Profile

Setting Up Your User Profile

Accessing Your Profile Click on your profile picture in the upper-right corner of the GitHub interface and select "Your profile" from the dropdown menu.

Editing Your Profile

1. On your profile page, click the "Edit profile" button.
2. Update your personal information, including your name, bio, location, and website URL.
3. Upload a profile picture to personalize your account.

Setting Up Your Profile Readme

1. Create a special repository named the same as your username (e.g., username/username).
2. Add a README.md file to this repository. The contents of this file will be displayed on your GitHub profile page.
3. Use Markdown to format your README, including text, images, links, and badges to showcase your projects and skills.

Customizing Profile Settings and Preferences

Accessing Settings Click on your profile picture in the upper-right corner and select "Settings" from the dropdown menu.

Personal Settings

1. Navigate to the "Account" section to update your account settings, including your email address and password.
2. In the "Security" section, enable two-factor authentication (2FA) for added security.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Notifications Go to the "Notifications" section to customize how you receive updates about activity on your repositories, issues, and pull requests.

Appearance Adjust the theme and layout of GitHub by visiting the "Appearance" section. Choose between light and dark modes according to your preference.

Setting Up SSH Keys for Authentication

What are SSH Keys and Why Use Them?

Definition SSH (Secure Shell) keys are a pair of cryptographic keys used to authenticate your computer to a remote server, such as GitHub, without using a password.

Benefits

- Enhanced security: SSH keys are more secure than traditional username/password authentication.
- Convenience: Once set up, SSH keys allow you to interact with GitHub without repeatedly entering your username and password.

Generating SSH Keys

Open a Terminal

- On Windows, you can use Git Bash, PowerShell, or Command Prompt.
- On macOS and Linux, use the built-in Terminal application.

Generate a New SSH Key Pair In the terminal, enter the following command:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

If your system doesn't support the ed25519 algorithm, you can use:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Follow the Prompts When prompted to "Enter a file in which to save the key," press Enter to accept the default file location. Enter a secure passphrase when prompted (optional, but recommended).

Add Your SSH Key to the SSH-Agent Start the SSH agent in the background:

```
eval "$(ssh-agent -s)"
```

Add your SSH private key to the SSH agent:

```
ssh-add ~/.ssh/id_ed25519
```

For RSA, the command would be:

```
ssh-add ~/.ssh/id_rsa
```

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Adding SSH Keys to Your GitHub Account

Copy the SSH Key to Your Clipboard Use the following command to copy the key:

```
clip < ~/.ssh/id_ed25519.pub
```

For RSA:

```
clip < ~/.ssh/id_rsa.pub
```

On macOS:

```
pbcopy < ~/.ssh/id_ed25519.pub
```

On Linux:

```
cat ~/.ssh/id_ed25519.pub
```

Then, manually copy the output.

Add the SSH Key to GitHub

1. In GitHub, click on your profile picture in the upper-right corner and select "Settings."
2. Navigate to "SSH and GPG keys" in the left sidebar.
3. Click the "New SSH key" button.
4. Provide a descriptive title for the key, paste your SSH key into the "Key" field, and click "Add SSH key."

Verify Your SSH Key Setup Test your SSH connection with the following command:

```
ssh -T git@github.com
```

If successful, you will see a message confirming that you have successfully authenticated.

By following these steps, you will have successfully created and configured your GitHub account, set up your user profile, and enabled SSH key authentication for secure and convenient interactions with GitHub.

Creating a New Repository

Creating a new repository on GitHub is straightforward and can be done directly through the GitHub interface.

Steps to Create a Repository from the GitHub Interface

Sign In to GitHub

1. Open your web browser and go to GitHub's website.
2. Sign in to your GitHub account using your username and password.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Navigate to the New Repository Page

1. Click on the "+" icon in the upper-right corner of the GitHub interface.
2. From the dropdown menu, select "New repository."

Fill Out Repository Details

- **Owner:** Select the owner of the repository. If you belong to an organization, you can choose the organization or your personal account.
- **Repository Name:** Enter a name for your repository. The name should be unique within the selected owner's repositories and relevant to your project.
- **Description (optional):** Provide a brief description of your repository. This is optional but recommended as it helps others understand the purpose of your project.

Choose Repository Visibility

- **Public:** Select this option if you want anyone to see your repository. Public repositories are ideal for open source projects.
- **Private:** Select this option if you want to restrict access to your repository. Private repositories are suitable for personal or proprietary projects.

Initialize the Repository

You have the option to initialize your repository with several default files. Initializing the repository with these files helps you get started quickly.

Initializing a Repository with a README, .gitignore, and License

Add a README File

- Check the box labeled "Add a README file."
- A README file is essential as it provides an overview of your project. You can edit the README file later to include detailed information about your project, usage instructions, and more.

Add a .gitignore File

- Check the box labeled "Add .gitignore."
- From the dropdown menu, select the type of project you are working on (e.g., Node, Python, Java). This selection will create a .gitignore file tailored to your project, which specifies which files and directories Git should ignore. This is useful for excluding files that are not meant to be tracked, such as build artifacts, dependencies, and environment-specific files.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Add a License

- Check the box labeled "Choose a license."
- From the dropdown menu, select a license that best suits your project. Common licenses include MIT, Apache 2.0, and GPL. Including a license file in your repository is important for open source projects as it defines the terms under which others can use, modify, and distribute your code.

Create Repository

Once you have filled out all the necessary details and selected your initialization options, click the "Create repository" button at the bottom of the page. GitHub will create your new repository and redirect you to its main page, where you can start managing your project.

Cloning a Repository from GitHub

Cloning a repository from GitHub allows you to create a local copy of the repository on your computer. This enables you to work on the project offline and use Git commands to manage changes.

Cloning Repositories to Your Local Machine

Sign In to GitHub

1. Open your web browser and go to GitHub's website.
2. Sign in to your GitHub account using your username and password.

Navigate to the Repository

1. Find the repository you want to clone. You can use the search bar at the top of the GitHub interface or navigate to the repository from your list of repositories.

Copy the Repository URL

1. On the main page of the repository, click the "Code" button located above the list of files.
2. Ensure you are on the "HTTPS" tab and click the clipboard icon to copy the repository URL. The URL will look something like this:

```
https://github.com/username/repository-name.git
```

Open a Terminal

Open your terminal application. On Windows, you can use Git Bash, Command Prompt, or PowerShell. On macOS and Linux, use the built-in Terminal application.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Navigate to Your Desired Directory

Use the `cd` command to navigate to the directory where you want to clone the repository. For example:

```
cd path/to/your/directory
```

Clone the Repository

Use the `git clone` command followed by the repository URL you copied earlier:

```
git clone https://github.com/username/repository-name.git
```

Git will create a new directory named after the repository and download all the files, branches, and commit history to your local machine.

Navigate to the Cloned Repository

Change into the newly created repository directory:

```
cd repository-name
```

Understanding the Structure of a Cloned Repository

Once you have cloned a repository, it's important to understand its structure and how Git organizes the files and directories:

Root Directory

The root directory of the cloned repository contains the main project files and directories. This is where you will find the `README.md`, `.gitignore`, and `LICENSE` files if they were initialized.

.git Directory

Inside the root directory, there is a hidden directory named `.git`. This directory contains all the metadata and history for the repository, including configurations, branches, and commit history. This directory is crucial for Git to track changes and manage the repository.

Project Files and Directories

The remaining files and directories are the actual contents of your project. These can include source code files, documentation, assets, configuration files, and more. The structure of these files will vary depending on the type of project.

Common Files

- **README.md:** A Markdown file that typically contains an introduction and instructions for the project. It's displayed on the repository's main page on GitHub.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

- **.gitignore:** A file that specifies which files and directories should be ignored by Git. This helps prevent unnecessary files (like build artifacts or temporary files) from being tracked.
- **LICENSE:** A file that defines the legal terms under which the project can be used, modified, and distributed.

Understanding Repository Settings

Configuring your repository settings is essential for managing your project effectively on GitHub.

Configuring Repository Settings (Description, Topics, Visibility)

Accessing Repository Settings

1. Navigate to the repository you want to configure.
2. Click on the "Settings" tab located on the right side of the repository's navigation bar.

Configuring the Repository Description

- In the "Options" section, you will find the "Repository name and description" area. Enter a brief and informative description of your repository. This helps others understand the purpose and scope of your project.
- Optionally, you can add a URL to the repository's homepage or documentation site.

Adding Topics to Your Repository

- Below the description, you can add topics (tags) to categorize your repository. Topics help improve discoverability on GitHub.
- To add topics, click on the "Manage topics" button, enter relevant keywords, and press Enter. Topics might include programming languages, frameworks, or specific technologies related to your project.

Configuring Repository Visibility

- Scroll down to the "Danger Zone" section to change the visibility of your repository.
- **Public Repository:** Select this option if you want anyone to see your repository. Public repositories are ideal for open-source projects.
- **Private Repository:** Select this option if you want to restrict access to your repository. Private repositories are suitable for personal or proprietary projects.
- To change the visibility, click on "Change repository visibility," select the desired option, and confirm the change.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Managing Collaborators and Permissions

Accessing Collaborators Settings

1. In the repository settings, click on the "Manage access" tab on the left sidebar.

Inviting Collaborators

1. To add collaborators to your repository, click on the "Invite a collaborator" button.
2. Enter the GitHub username or email address of the person you want to invite.
3. Select the appropriate permission level for the collaborator:
    - **Read:** Allows the collaborator to view and clone the repository.
    - **Triage:** Allows the collaborator to manage issues and pull requests.
    - **Write:** Allows the collaborator to push commits to the repository and manage issues and pull requests.
    - **Maintain:** Allows the collaborator to manage repository settings, including adding collaborators.
    - **Admin:** Grants full access to the repository, including the ability to delete the repository.
4. Click "Add" to send the invitation.

Managing Existing Collaborators

- In the "Manage access" section, you can see the list of current collaborators and their permission levels.
- To change a collaborator's permission level, click on the "Edit" button next to their name and select the new permission level.
- To remove a collaborator, click on the "Remove" button next to their name.

Forking Repositories

Forking a repository is a common practice on GitHub, especially in open-source projects. It allows you to create your own copy of someone else's repository so you can make changes without affecting the original project.

Why Use Forking?

- Forking allows you to experiment with changes without affecting the original repository. You can try new ideas, build features, or test bug fixes in your forked copy.
- When you fork a repository, you can contribute to the original project by submitting pull requests. This is a common workflow in open-source projects where multiple developers collaborate.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

- You can customize the project to meet your specific needs or integrate it into your own projects while still being able to pull in updates from the original repository.
- Forking a repository is a great way to learn from other developers' code. You can explore how a project is structured and try making your own improvements.

How to Fork a Repository

Sign In to GitHub

1. Open your web browser and go to GitHub's website.
2. Sign in to your GitHub account using your username and password.

Navigate to the Repository

Find the repository you want to fork. You can use the search bar at the top of the GitHub interface or navigate to the repository from another user's profile or an organization's page.

Fork the Repository

1. On the repository's main page, click the "Fork" button located in the upper-right corner of the page. The button is typically next to the "Star" button.
2. GitHub will ask you where you want to fork the repository. Choose your personal account or one of your organizations.
3. GitHub will create a copy of the repository under your account. This process might take a few moments, depending on the size of the repository.

Access Your Forked Repository

Once the forking process is complete, GitHub will redirect you to the newly forked repository under your account. The URL will look something like this:

```
https://github.com/your-username/forked-repository
```

Clone Your Forked Repository (Optional)

To start working on your forked repository locally, you need to clone it to your machine.

1. Copy the URL of your forked repository from the "Code" button.
2. Open your terminal application and navigate to the directory where you want to clone the repository.
3. Use the `git clone` command followed by the repository URL:

   ```
   git clone https://github.com/your-username/forked-repository.git
   ```

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Making Changes and Contributing

You can now make changes to your forked repository. Once you're ready to contribute your changes back to the original repository, you can create a pull request.

To do this, push your changes to a branch in your forked repository and then navigate to the original repository. GitHub will suggest creating a pull request from your branch.

Making Pull Requests

Pull requests are a core feature of collaborative development on GitHub. They allow you to propose changes to a repository, discuss them with the repository owners and contributors, and ultimately have your changes merged into the main project.

What are Pull Requests?

- A pull request (PR) is a method of submitting contributions to a project. It lets you notify project maintainers about changes you've pushed to a branch in your fork or branch of the repository.
- Pull requests facilitate code review and discussion. Other developers can see your changes, provide feedback, suggest improvements, and approve or request changes before your code is merged.

Purpose of Pull Requests

- Pull requests allow for peer review of code changes, ensuring code quality and consistency.
- They provide a platform for collaboration and discussion, making it easier for multiple developers to work together on a project.
- Pull requests keep a record of changes and discussions, which is useful for future reference and understanding the evolution of the project.

Creating and Submitting a Pull Request

Make Changes on a Branch

Before creating a pull request, make sure your changes are on a separate branch. This branch should be based on the branch you want to merge into (e.g., main or develop).

```
git checkout -b my-feature-branch
# Make your changes
git add .
git commit -m "Description of the changes"
```

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Push Your Branch to GitHub

Push your branch to your fork or the repository:

```
git push origin my-feature-branch
```

Navigate to the Repository on GitHub

Go to the repository on GitHub where you want to submit the pull request. This can be either your fork or the original repository.

Initiate a Pull Request

- If your branch is in your fork, navigate to the original repository and GitHub will prompt you to create a pull request from your recent push. Click the "Compare pull request" button.
- If you are on the repository page, click the "Pull requests" tab, then click the "New pull request" button.
- Select the branch you want to merge into (e.g., main) and the branch with your changes (e.g., my-feature-branch).

Fill Out the Pull Request Form

- **Title:** Provide a clear and concise title for your pull request.
- **Description:** Write a detailed description of the changes you've made. Explain why the changes are necessary, what problem they solve, and any other relevant context.

Submit the Pull Request

Once you've filled out the form, click the "Create pull request" button to submit your pull request. Your pull request will be listed in the "Pull requests" tab of the repository, where maintainers and collaborators can review and discuss it.

Best Practices for Pull Request Descriptions

- Write a clear and concise title and description. The title should summarize the changes, and the description should provide detailed context.
- Explain what changes you made and why. Include details about the problem you are solving or the feature you are adding.
- If your pull request addresses an issue, reference it in the description using the issue number (e.g., "Fixes #123"). This links the pull request to the issue and provides additional context.
- Provide any relevant background information or context that will help reviewers understand your changes. This might include design decisions, screenshots, or performance implications.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

- Highlight the most important changes, especially if your pull request is large. This helps reviewers focus on the critical parts of your submission.
- If you need feedback on particular aspects of your changes, mention this in the description. Directing reviewers' attention to specific areas can make the review process more efficient.
- Describe how you tested your changes and any steps reviewers should take to verify them. This can include automated tests, manual testing steps, or screenshots of the changes in action.

Reviewing and Merging Pull Requests

Effective review and merging of pull requests (PRs) are critical to maintaining the quality and stability of a codebase.

Accessing the Pull Request

1. Navigate to the repository on GitHub.
2. Click on the "Pull requests" tab to see a list of open PRs.
3. Select the pull request you want to review.

Understanding the Changes

- On the pull request page, you will see a summary of the changes, including the title, description, and any linked issues.
- Review the description to understand the purpose and context of the changes.

Viewing the Code Changes

1. Click on the "Files changed" tab to see the modified files and lines of code.
2. GitHub will show a diff view, highlighting additions in green and deletions in red.
3. Carefully review the changes, looking for potential issues, improvements, or inconsistencies.

Discussing and Making Code Review Comments

Adding Comments

1. To comment on a specific line of code, click on the "+" icon next to the line number.
2. Enter your feedback, suggestions, or questions in the comment box and click "Start a review" or "Add single comment."
3. Use inline comments to provide precise feedback on specific parts of the code.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Starting a Review

1. If you have multiple comments, click "Start a review" for the first comment. This allows you to submit all comments together.
2. Continue adding comments as needed. They will be collected into a single review.

Submitting Your Review

1. Once you have finished adding comments, click the "Review changes" button at the top right.
2. Select the appropriate option:
    - Comment: Submit your comments without approving or requesting changes.
    - Approve: Approve the changes if you find them satisfactory.
    - Request changes: Request changes if there are issues that need to be addressed before merging.
3. Click "Submit review" to finalize your feedback.

Engaging in Discussion

- Engage in discussions with the PR author and other reviewers. Use the comment threads to clarify questions, discuss alternatives, and iterate on the code changes.
- Be respectful and constructive in your feedback, aiming to improve the code quality and facilitate learning.

Merging Pull Requests and Resolving Conflicts

Merging the Pull Request

1. Once the pull request has been reviewed and approved, it can be merged into the target branch.
2. Navigate to the bottom of the PR page and click the "Merge pull request" button.
3. Choose the merge method:
    - Create a merge commit: Keeps all commits from the feature branch.
    - Squash and merge: Combines all commits into a single commit.
    - Rebase and merge: Reapplies commits from the feature branch onto the base branch.
4. Enter a commit message if required and click "Confirm merge."

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Resolving Merge Conflicts

Sometimes, conflicts arise when the changes in the PR conflict with the target branch. GitHub will alert you if there are merge conflicts.

1. Click the "Resolve conflicts" button.
2. GitHub will show the conflicting files and sections.
3. Manually edit the files to resolve the conflicts. Look for markers like `<<<<<<`, `======`, and `>>>>>>` to identify conflicting sections.
4. After resolving conflicts, mark the file as resolved and commit the changes.

Finalizing the Merge

1. After resolving conflicts, you may need to re-review the PR to ensure the conflicts were resolved correctly.
2. Once the PR is ready, repeat the merging process to merge the PR into the target branch.

Issues and Project Boards

Effectively managing issues and organizing work with project boards are crucial for maintaining a well-structured and productive project.

Creating an Issue

1. Navigate to the repository on GitHub.
2. Click on the "Issues" tab.
3. Click the "New issue" button to open the issue creation form.
4. **Title:** Provide a concise and descriptive title for the issue.
5. **Description:** Write a detailed description of the issue, including any relevant information such as steps to reproduce a bug, expected behavior, actual behavior, and any screenshots or logs.
6. **Templates:** Use issue templates if available. Templates provide a structured format for reporting bugs, requesting features, or submitting other types of issues.

Submitting the Issue

After filling out the issue details, click the "Submit new issue" button. The issue will be added to the list of open issues for the repository, where it can be tracked and managed.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Managing Issues

1. To update an issue, navigate to the issue and click the "Edit" button.
2. You can comment on issues to provide updates, ask for additional information, or discuss potential solutions.
3. Close an issue when it has been resolved by clicking the "Close issue" button. You can also close issues automatically by including keywords like "Fixes #123" in your commit messages.

Using Labels, Milestones, and Assignees

Labels are tags that help categorize and prioritize issues.

- **Creating Labels:** Navigate to the "Labels" section under the "Issues" tab. Click "New label" to create a label, provide a name and description, and choose a color.
- **Applying Labels:** Open an issue and click the "Labels" button on the right sidebar. Select the relevant labels to apply them to the issue.

Milestones are used to group issues that are related to a specific goal or timeframe.

- **Creating Milestones:** Navigate to the "Milestones" section under the "Issues" tab. Click "New milestone," provide a title, description, and due date (optional), and click "Create milestone."
- **Assigning Issues to Milestones:** Open an issue and click the "Milestone" button on the right sidebar. Select the relevant milestone to associate it with the issue.

Assignees are team members responsible for addressing an issue.

- **Assigning Issues:** Open an issue and click the "Assignees" button on the right sidebar. Select one or more team members to assign them to the issue.

Organizing Work with Project Boards (Kanban Style)

Creating a Project Board

1. Navigate to the repository on GitHub.
2. Click on the "Projects" tab.
3. Click the "New project" button to create a new project board.
4. **Project Name:** Provide a name for the project.
5. **Template:** Choose a template, such as "Basic Kanban," which includes columns for "To do," "In progress," and "Done."
6. Click "Create project" to set up the board.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Adding Columns

Customize your project board by adding or renaming columns. Click "Add column" to create new columns as needed. Common columns include "Backlog," "Review," and "Testing."

Adding Issues to the Project Board

- To add issues to the project board, navigate to the "Issues" tab.
- Open the issue you want to add and click the "Projects" button on the right sidebar. Select the relevant project board and the column to place the issue in.

Managing Issues on the Project Board

- Drag and drop issues between columns to update their status. For example, move an issue from "To do" to "In progress" when work begins.
- Use the board to visualize the workflow, track progress, and ensure that work is moving forward efficiently.

Tracking Progress with Project Boards

- Use the project board to monitor the overall progress of your project. Columns give a clear view of which tasks are pending, in progress, or completed.
- Update issues and move them across columns as work progresses to keep the project board up to date.

By using issues, labels, milestones, assignees, and project boards effectively, you can enhance project management, improve team collaboration, and maintain a clear and organized workflow. This structured approach helps ensure that tasks are tracked, prioritized, and completed efficiently.

Overview of GitHub Pages

GitHub Pages is a static site hosting service provided by GitHub. It allows you to host web pages directly from a GitHub repository. These pages can be HTML, CSS, JavaScript, or generated from a variety of static site generators such as Jekyll, Hugo, or others. The service is free and can be used to host personal, project, or organization sites.

Key Features of GitHub Pages

- Hosting a website on GitHub Pages is straightforward and involves pushing your web files to a GitHub repository.
- You can configure custom domains for your GitHub Pages site.
- GitHub Pages supports HTTPS for secure connections, and GitHub automatically provides SSL certificates for custom domains.
- Changes to your website can be made via commits to your repository, fitting seamlessly into your Git workflow.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Use Cases for GitHub Pages

GitHub Pages is versatile and can be used for various purposes. Here are some common use cases:

Personal Websites and Portfolios

- Developers, designers, and other professionals use GitHub Pages to create personal websites and portfolios. This is an excellent way to showcase projects, skills, and achievements.
- You can host an online resume or CV, making it easily accessible to potential employers and collaborators.

Project Documentation

- Many open-source projects use GitHub Pages to host detailed user guides, manuals, and API documentation. This makes it easier for users and contributors to understand and use the project.
- GitHub Pages can serve as a wiki or knowledge base for a project, providing a centralized location for information and resources.

Blogs and Technical Writing

- Developers often use GitHub Pages to host personal blogs where they share tutorials, technical articles, and thoughts on various topics.
- Technical writers and educators use GitHub Pages to create and share tutorials, course materials, and other educational content.

Organization and Community Sites

- Communities and organizations use GitHub Pages to create websites that serve as hubs for their activities, announcements, and resources.
- GitHub Pages can be used to create websites for events, such as conferences, meetups, and hackathons, providing information about schedules, speakers, and registration.

Project Showcases

- Developers use GitHub Pages to create demo sites that showcase the features and functionality of their projects. This is particularly useful for front-end projects and libraries.
- Projects can have dedicated landing pages that provide an overview of the project, its goals, and links to relevant resources.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Creating and Deploying a GitHub Pages Site

Setting up a GitHub Pages site involves configuring a repository, choosing a publishing source, and customizing the deployed site. Here's a step-by-step guide to help you through the process:

Setting Up a Repository for GitHub Pages

Create a New Repository

1. Sign in to your GitHub account.
2. Click the "+" icon in the upper-right corner of the page and select "New repository."
3. Name your repository. For a user or organization site, name it `<username>.github.io` (e.g., `myusername.github.io`). For project sites, you can use any name.
4. Optionally, add a description.
5. Choose the repository visibility (public or private).
6. Initialize the repository with a README file if desired.
7. Click "Create repository."

Add Web Content

1. Clone the repository to your local machine:

```
git clone https://github.com/yourusername/your-repo-name.git
cd your-repo-name
```

2. Add your website files to the repository. These can include HTML, CSS, JavaScript, images, and any other assets.
3. Commit and push the changes:

```
git add .
git commit -m "Initial commit"
git push origin main
```

Choosing a Publishing Source

Navigate to Repository Settings

1. Go to your repository on GitHub.
2. Click on the "Settings" tab at the top of the repository page.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Configure GitHub Pages

1. Scroll down to the "GitHub Pages" section.
2. Under "Source," select the branch you want to use for your site. Common choices are:
    - `main`: This is the default branch for most repositories.
    - `gh-pages`: A dedicated branch for GitHub Pages.
3. Optionally, select a folder within the branch, such as `/docs`, if your site files are located in a subdirectory.

Save Settings

After selecting the source branch and folder, click "Save." GitHub will build and deploy your site. This may take a few moments.

Deploying and Customizing Your GitHub Pages Site

Access Your Site

Once the deployment is complete, your site will be available at `https://<username>.github.io/<repos` or `https://<username>.github.io` for user/organization sites. You can find the URL in the "GitHub Pages" section of the repository settings.

Customizing Your Site

- GitHub Pages offers built-in themes that you can apply to your site. In the "GitHub Pages" section, click "Choose a theme" and select a theme. This will create a `_config.yml` file in your repository with theme configuration.
- To use a custom domain, go to the "GitHub Pages" section and enter your custom domain in the "Custom domain" field. Create a DNS record with your domain registrar pointing to GitHub's servers (specific instructions are provided in the GitHub documentation).
- GitHub Pages natively supports Jekyll, a static site generator. You can use Jekyll to add dynamic content, templates, and layouts. Create a `_config.yml` file in the root of your repository and add Jekyll configuration. Organize your content using Jekyll's conventions (e.g., `_posts` for blog posts).

Updating Your Site

To update your site, simply push new commits to the branch you configured as the source for GitHub Pages. GitHub will automatically rebuild and deploy your site whenever new changes are pushed.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.

Troubleshooting

- If your site fails to build, GitHub will display error messages in the "GitHub Pages" section. Review these messages to diagnose and fix the issues.
- Sometimes, changes may not appear immediately due to caching. Clear your browser cache or wait for the cache to refresh.

By following these steps, you can create, deploy, and customize a GitHub Pages site effectively. GitHub Pages provides a straightforward way to host static websites, making it an excellent choice for personal portfolios, project documentation, and more.

Copyright © 2024 John Scott Tremaine. Content for BreakpointCoding.com. Not for redistribution.