What is GitHub?

The GitHub flow

In addition to providing a platform for collaborative software development, GitHub also offers a workflow designed to optimize use of its various features. While this unit offers a cursory overview of important platform components, it's recommended that you first review Understanding the GitHub flow.

Git and GitHub

As you work with **Git** and **GitHub**, you may wonder about the difference between the two.

Git is a distributed version control system (DVCS) that allows multiple developers or other contributors to work on a project. It provides a way to work with one or more local branches and push them to a remote repository. Git is responsible for everything GitHub-related that happens locally on your computer. Key features provided by Git include:

- Installed and used on your local machine
- Handles version control
- Supports branching

To learn more about **Git**, see Using common Git commands.

GitHub is a cloud platform that uses Git as its core technology. It simplifies the process of collaborating on projects and provides a website, command-line tools, and overall flow that allows developers and users to work together. GitHub acts as the "remote repository" mentioned previously in the **Git** section.

Key features provided by GitHub include:

- Issues
- Discussions
- Pull requests
- Notifications
- Labels
- Actions
- Forks
- Projects

To learn more about **GitHub**, see Getting started with GitHub.

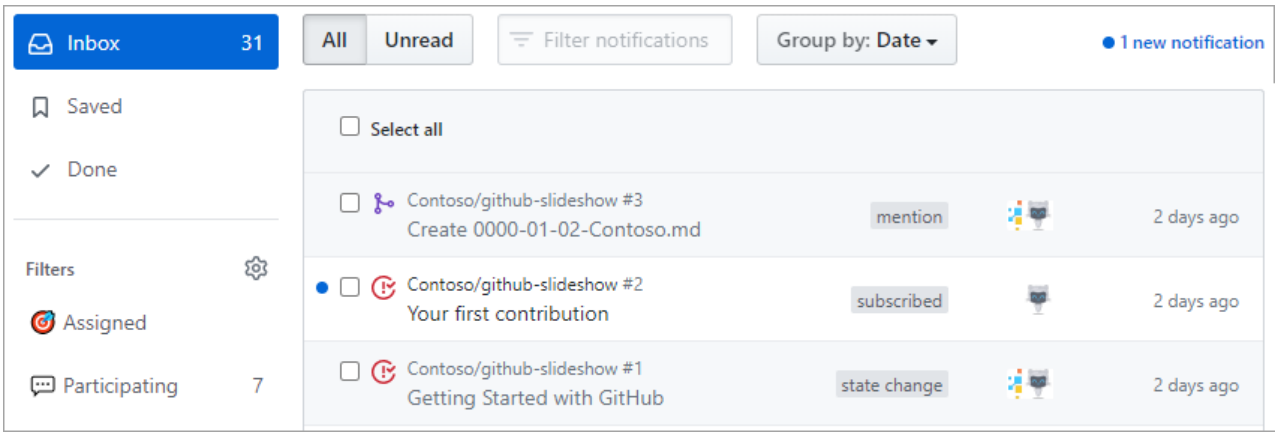
Issues

Issues are where most of the communication between a project's consumers and development team occurs. An *issue* can be created to discuss a broad set of topics, including bug reports, feature requests, documentation clarifications, and more. Once an issue has been created, it can be assigned to owners, labels, projects, and milestones. You can also associate issues with pull requests and other GitHub items to provide future traceability.

To learn more about GitHub Issues, see Mastering Issues.

Notifications

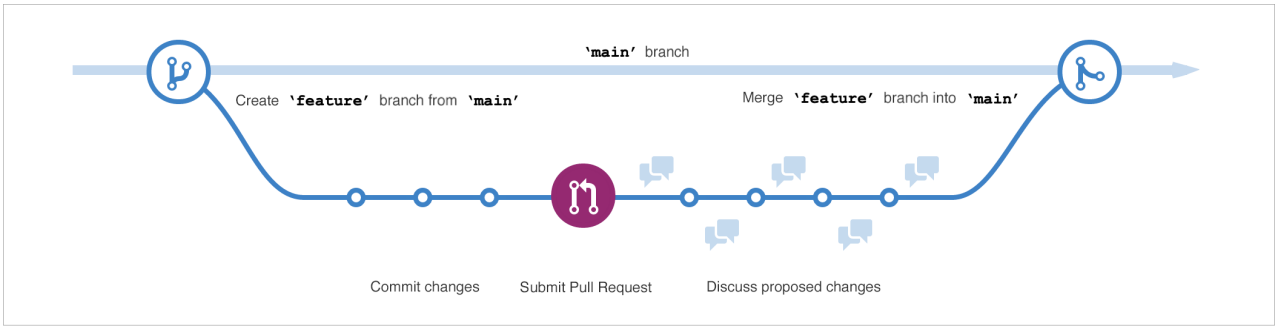
As a collaborative platform, GitHub offers **notifications** for virtually every event that takes place within a given workflow. These notifications can be finely tuned to meet your preferences. For example, you can subscribe to all issue creations and edits on a project, or you can just receive notifications for issues in which you are mentioned. You can also decide whether you receive notifications via email, web & mobile, or both. To keep track of all of your notifications across different projects, use the GitHub Notifications dashboard.



To learn more about GitHub notifications, see Configuring notifications.

Branches

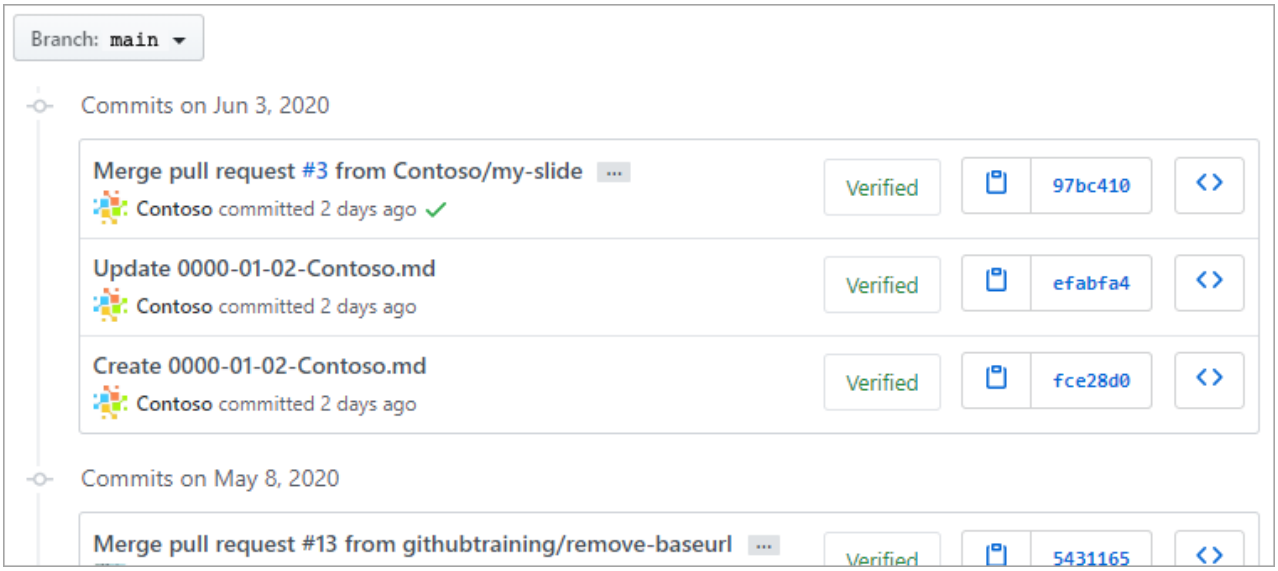
Branches are the preferred way to create changes in the GitHub flow. They provide isolation so that multiple people may simultaneously work on the same code in a controlled way. This model enables stability among critical branches, such as `main`, while allowing complete freedom for developers to commit any changes they need to meet their goals. Once the code from a branch is ready to become part of the `main` branch, it may be merged via pull request.



To learn more about GitHub branches, see About branches.

Commits

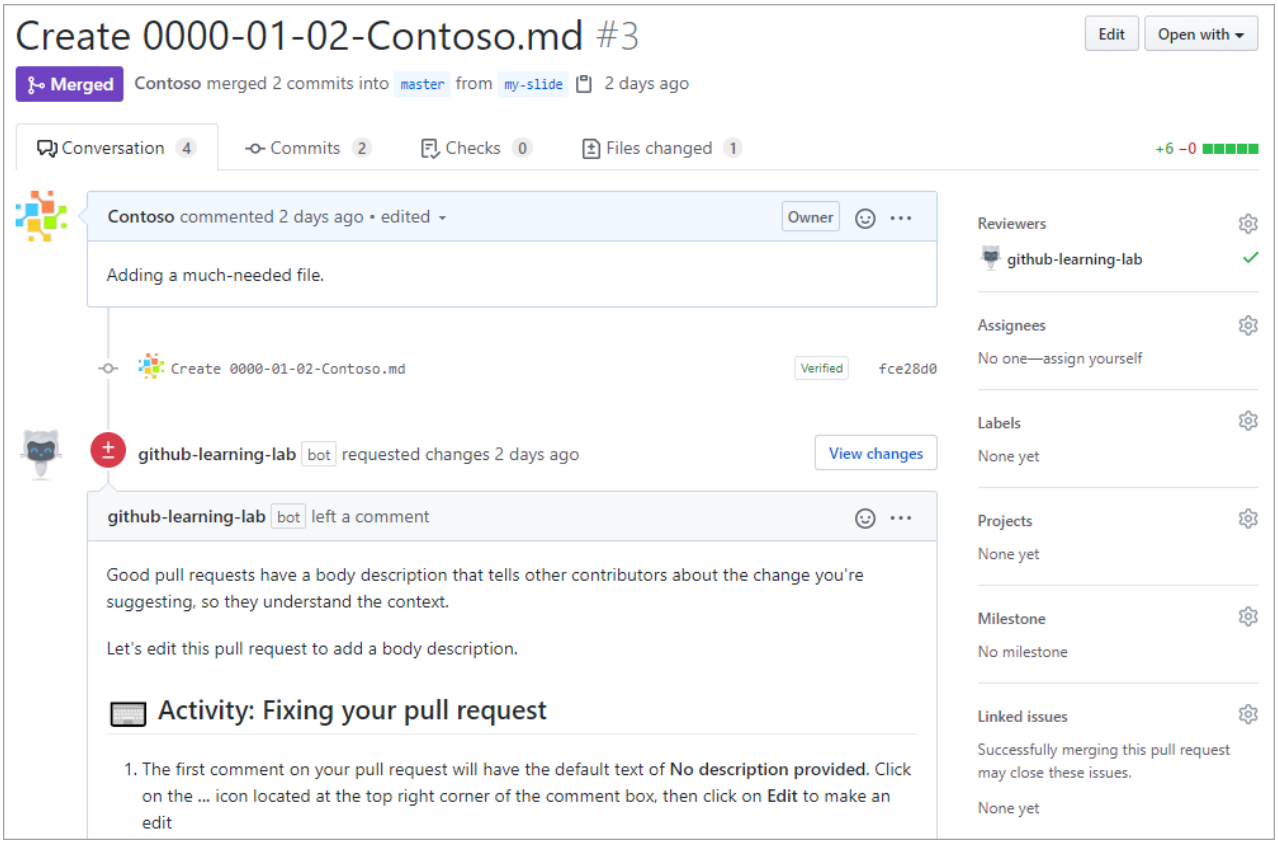
A **commit** is a change to one or more files on a branch. Every time a commit is created, it is assigned a unique ID and tracked, along with the time and contributor. This provides a clear audit trail for anyone reviewing the history of a file or linked item, such as an issue or pull request.



To learn more about GitHub commits, see Committing and reviewing changes to your project.

Pull Requests

A **pull request** is the mechanism used to signal that the commits from one branch are ready to be merged into another branch. The developer submitting the **pull request** will often request one or more reviewers to verify the code and approve the merge. These reviewers have the opportunity to comment on changes, add their own, or use the pull request itself for further discussion. Once the changes have been approved (if approval is required), the pull request's source branch (the compare branch) may be merged in to the base branch.



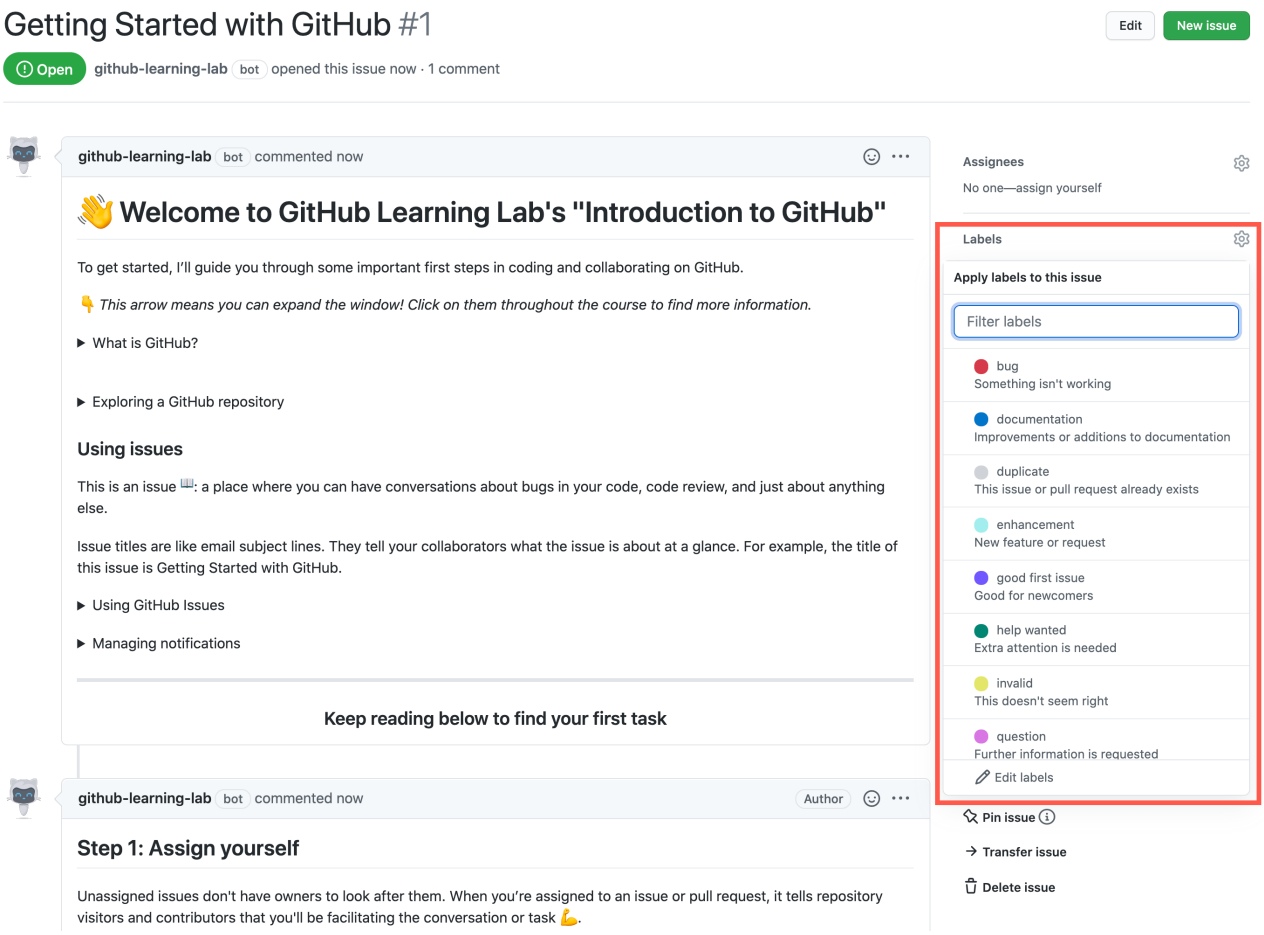
To learn more about GitHub pull requests, see About pull requests.

Labels

Labels provide a way to categorize and organize **issues** and **pull requests** in a repository. As you create a GitHub repository several labels will automatically be added for you and new ones can also be created.

Examples of Labels include:

- bug
- documentation
- duplicate
- help wanted
- enhancement
- question



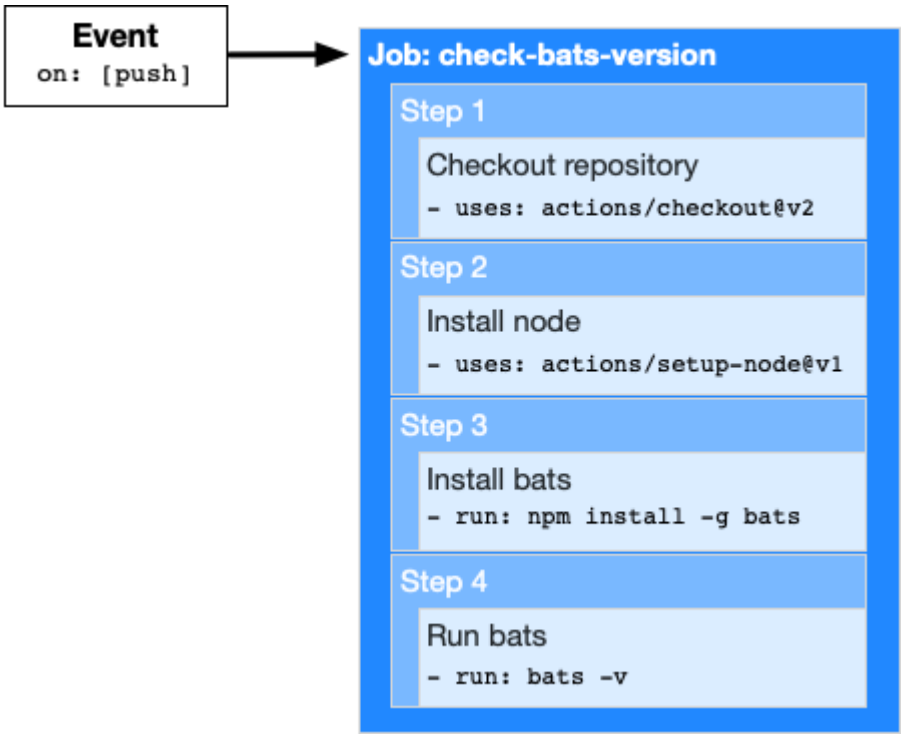
To learn more about GitHub labels see About labels.

Actions

GitHub actions provide task automation and workflow functionality in a repository. Actions can be used to streamline processes in your software development lifecycle and implement continuous integration and continuous deployment (CI/CD).

GitHub Actions are composed of the following components:

- **Workflows**: Automated processes added to your repository.
- **Events**: An activity that triggers a workflow.
- **Jobs**: A set of steps that execute on a runner.
- **Steps**: A task that can run one or more commands (actions).
- **Actions**: Standalone commands that can be combined into steps. Multiple steps can be combined to create a job.
- **Runners**: Server that has the GitHub Actions runner application installed.



To learn more about GitHub actions see Introduction to GitHub Actions.

Cloning and forking

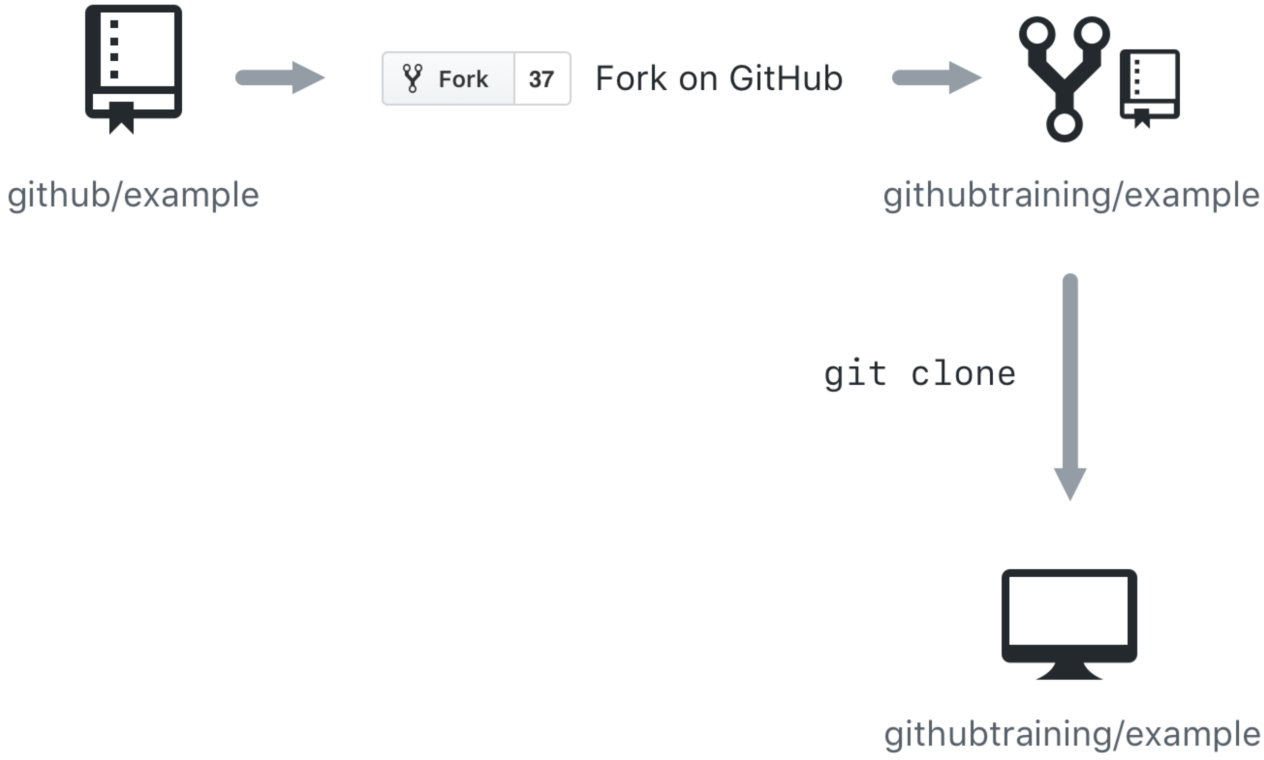
GitHub provides multiple ways to copy a repository so that you can work on it.

- **Cloning a Repository** - Cloning a repository will make a copy of the repository and its history on your local machine. If you have write access to the repository you can push changes from your local machine to the remote repository (called the **origin**) as they're completed. To clone a repository you can use the `git clone [url]` command or the GitHub CLI's `gh repo clone [url]` command.
- **Forking a Repository** - **Forking** a repository makes a copy of the repository in your GitHub account. The parent repository is referred to as the **upstream** while your forked copy is referred to as the **origin**. Once you've forked a repository into your GitHub account you can **clone** it to your local machine. Forking allows you to freely make changes to a project without affecting the original **upstream** repository. To contribute changes back to the **upstream** repository you create a **pull request** from your forked repository. You can also run `git` commands to ensure that your local copy stays synced with the **upstream** repository.

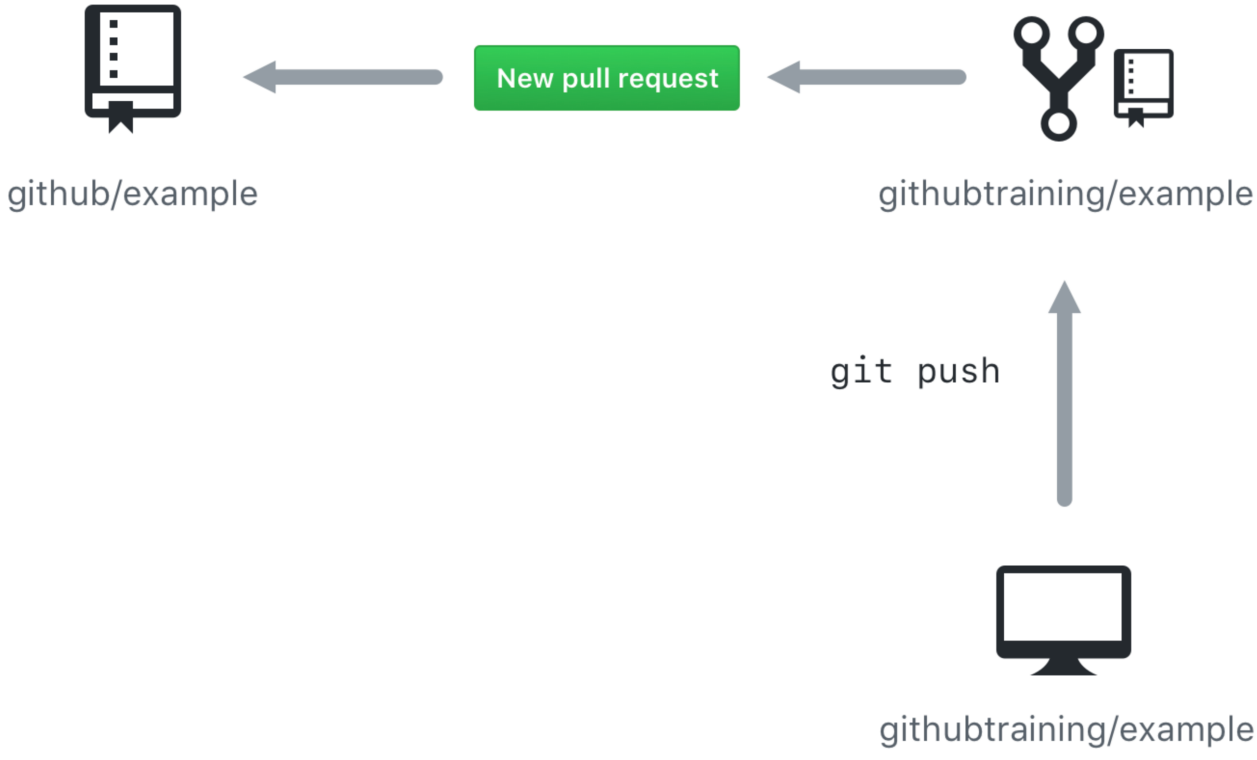
When would you clone a repository versus fork a repository? If you're working with a repository and have write access you can clone it to your local machine. From there you can make modifications and push your changes directly to the **origin** repository.

If you need to work with a repository created by another owner such as `github/example` and don't have write access, you can fork the repository into your GitHub account, and then clone the fork to your local machine. To see this visually, let's assume that your GitHub account is called `githubtraining` for this example. Using the

GitHub website you can fork `githubtraining` or any other examples into your account. From there you can clone the forked version of the repository to your local machine. These steps are shown in the following image.



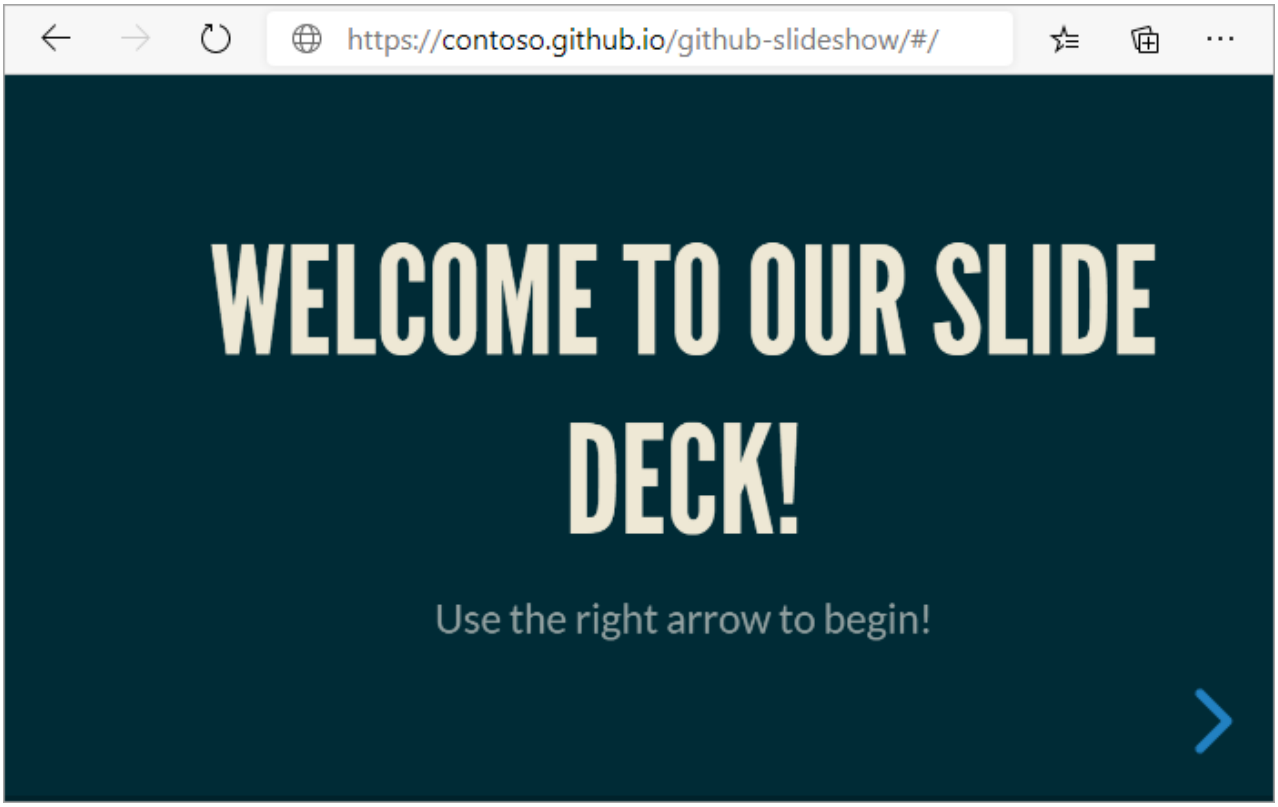
Changes can be made to your local copy of `githubtraining/example` and then pushed back to your remote **origin** repository (`githubtraining/example`). The changes can then be submitted to the `github/example` **upstream** repository using a **pull request** as shown next.



To learn more, see Fork a repo.

GitHub Pages

GitHub Pages is a hosting engine that's built right into your GitHub account. By following a few conventions, and enabling the feature, you can build your own static site generated from HTML and markdown code pulled directly from your repository.



To learn more, see GitHub Pages.

Next unit: Exercise - A guided tour of GitHub

Continue