Contents

[Starting Time 1](#_Toc460258342)

[Understanding the GitHub Flow 1](#_Toc460258343)

[Create a branch 1](#_Toc460258344)

# Starting Time

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | August | | | | | | | | --- | --- | --- | --- | --- | --- | --- | | M | T | W | T | F | S | S | |  | 1 | 2 | 3 | 4 | 5 | 6 | | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | 14 | 15 | 16 | 17 | 18 | 19 | 20 | | 21 | 22 | 23 | 24 | 25 | 26 | 27 | | 28 | **29** | 30 | 31 |  |  |  | | 6:22 PM |

# Understanding the GitHub Flow

GitHub Flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. This guide explains how and why GitHub Flow works.

# Create a branch

When you’re working on a project, you’re going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

When you create a branch in your project, you’re creating an environment where you can try out new ideas. Changes you make on a branch don’t affect the master branch, so you’re free to experiment and commit changes, safe in the knowledge (15 minutes) that your branch won’t be merged until it’s ready to be reviewed by someone you’re collaborating with.

## ProTip

Branching is a core concept in Git, and the entire GitHub Flow is based upon it. There’s only one rule: anything in the master branch is always deployable.

Because of this, it’s extremely important that your new branch is created off of master when working on a feature or a fix. Your branch name should be descriptive (e.g., refactor-authentication, user-content-cache-key, make-retina-avatars), so that others can see what is being worked on.

# Add commits

Once your branch has been created, it’s time to start making changes. Whenever you add, edit or delete a file, you’re making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

Commits also create a transparent history of your work that others can follow to understand what you’ve done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. Furthermore, each commit is considered a separate unit of change. This lets you roll back changes if a bug is found, or if you decide to head in a different direction. (15 minutes)

## ProTip

Commit messages are important, especially since Git tracks your changes and then displays them as commits once they’re pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback. (15 minutes - english).

# Open a Pull Request

Pull Requests initiate discussion about your commits. Because they’re tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process: when you have little or no code but want to share some screenshots or general ideas, when you’re stuck and need help or advice, or when you’re ready for someone to review your work. By using GitHub’s @mention system in your Pull Request message, you can ask for feedback from specific people or teams, whether they’re down the hall or ten time zones away. (15 minutes)

## ProTip

Pull Requests are useful for contributing to open source projects and for managing changes to shared repositories. If you’re using a Fork & Pull Model, Pull Requests provide a way to notify project maintainers about the changes you’d like them to consider. If you’re using a Shared Repository Model, Pull Requests help start code review and conversation about proposed changes before theyre merged into the master branch.

# Discuss and review your code

Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn’t match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation.

You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. GitHub will show your new commits and any additional feedback you may receive in the unified Pull Request view. (15 minutes)

ProTip

Pull Request comments are written in Markdown, so you can embed images and emoji, use pre-formatted text blocks, and other lightweight formatting.

# Deploy

Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them in production. If your branch causes issues, you can roll it back by deploying the existing master into production.

# Merge

Now that your changes have been verified in production, it is time to merge your code into the master branch.

Once merged, Pull Requests preserve a record of the historical changes to your code. Because they’re searchable, they let anyone go back in time to understand why and how a decision was made.

# ProTip

By incorporating certain keywords into the text or your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issued are also closed. For example, entering the phrase Closes #32 would close issue number 32 in the repository. (11 minutes 19 seconds)