

1. Introduction to Git

- What is Git?
 - A distributed version-control system for tracking changes in source code during software development.
 - It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

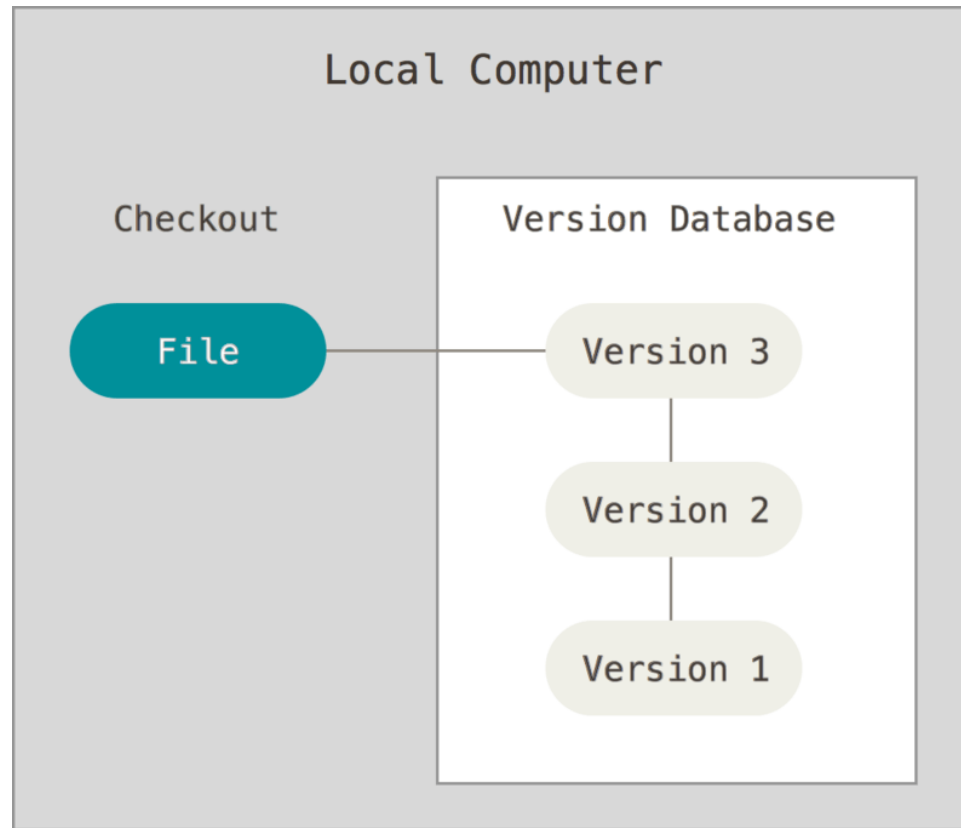


1. Introduction to Git

- What is version control?
 - A system that records changes to a file or set of files over time so that you can recall specific versions later.
 - It allows you to
 - revert selected files or the entire project back to a previous state,
 - compare changes over time,
 - see who last modified something that might be causing a problem,
 - who introduced an issue and when, and more.

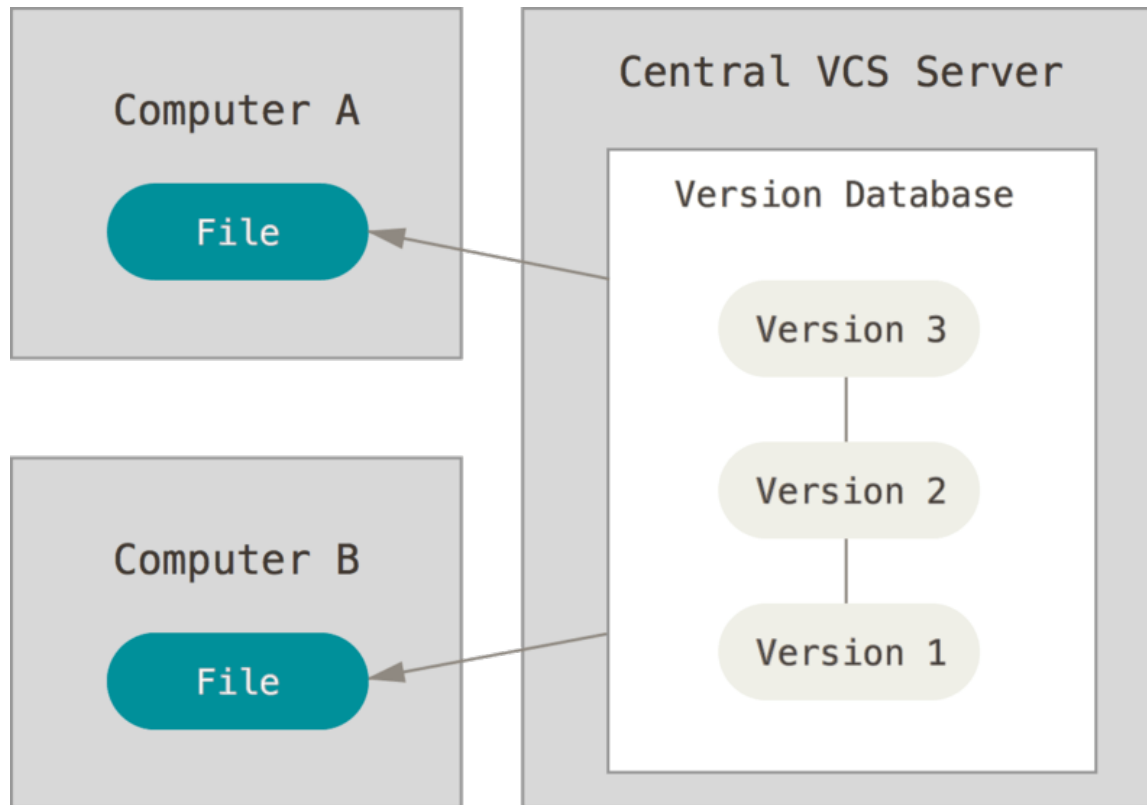
1. Introduction to Git

- Local Version Control System



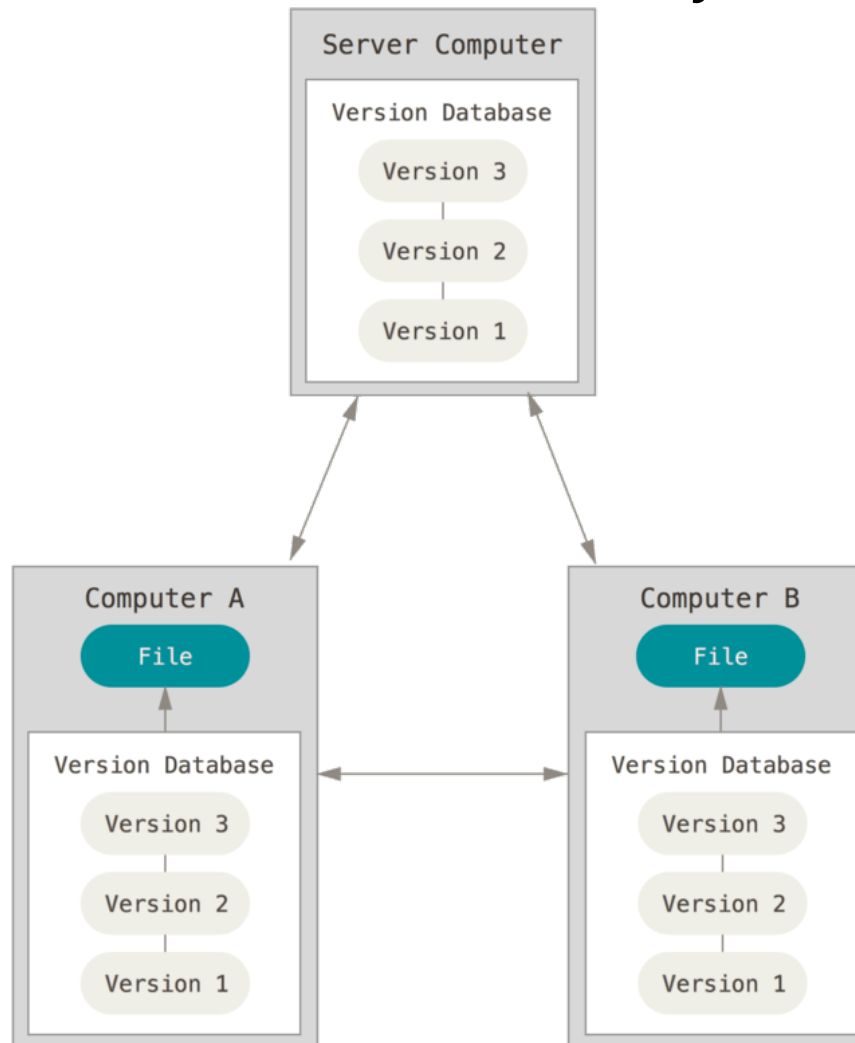
1. Introduction to Git

- Centralized Version Control System



1. Introduction to Git

- Distributed Version Control System



2. Create a GitHub account

What is GitHub?

<https://youtu.be/noZnOSpcjYY>

- It offers the distributed version control and source code management functionality of Git.

Create a GitHub account

- <https://github.com/join>

2. GitHub – Hello World

Lab exercise #1: <https://guides.github.com/activities/hello-world/>

1. Create a repository
2. Create a branch
3. Make and commit changes
4. Open a pull request
5. Merge your pull request
6. Fill in your info in the following spreadsheet by 5pm on Sep. 10th

https://docs.google.com/spreadsheets/d/1ZPvyk84a_k4dzPP8NCv4uRJqSABV4K9PUCfdtgfKhvU/edit?usp=sharing

2. GitHub – Hello World

Step 1. Create a Repository

A **repository** is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a *README*, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. *It also offers other common options such as a license file.*

Your `hello-world` repository can be a place where you store ideas, resources, or even share and discuss things with others.

2. GitHub – Hello World

Step 1. Create a Repository

To create a new repository

1. In the upper right corner, next to your avatar or identicon, click + and then select **New repository**.
2. Name your repository `hello-world`.
3. Write a short description.
4. Select **Initialize this repository with a README**.

2. GitHub – Hello World

Step 2. Create a Branch

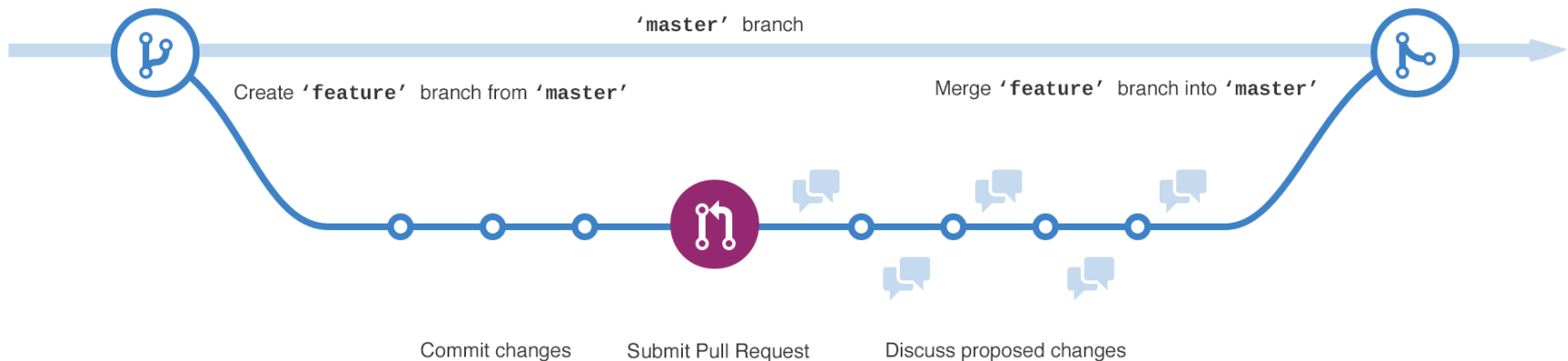
Branching is the way to work on different versions of a repository at one time.

By default your repository has one branch named `main` which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to `main`.

When you create a branch off the `main` branch, you're making a copy, or snapshot, of `main` as it was at that point in time. If someone else made changes to the `main` branch while you were working on your branch, you could pull in those updates.

2. GitHub – Hello World

Step 2. Create a Branch



This diagram shows:

- The `main` branch
- A new branch called `feature` (because we're doing 'feature work' on this branch)
- The journey that `feature` takes before it's merged into `main`

2. GitHub – Hello World

Step 2. Create a Branch

To create a new branch

1. Go to your new repository `hello-world`.
2. Click the drop down at the top of the file list that says **branch: main**.
3. Type a branch name, `readme-edits`, into the new branch text box.
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.

2. GitHub – Hello World

Step 3. Make and commit changes


Bravo! Now, you're on the code view for your `readme-edits` branch, which is a copy of `main`. Let's make some edits.

On GitHub, saved changes are called *commits*. Each commit has an associated *commit message*, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why.

2. GitHub – Hello World

Step 3. Make and commit changes

Make and commit changes

1. Click the `README.md` file.
2. Click the  pencil icon in the upper right corner of the file view to edit.
3. In the editor, write a bit about yourself.
4. Write a commit message that describes your changes.
5. Click **Commit changes** button.

2. GitHub – Hello World

Step 4. Open a Pull Request

Nice edits! Now that you have changes in a branch off of `main`, you can open a *pull request*.

Pull Requests are the heart of collaboration on GitHub. When you open a *pull request*, you're proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show *diffs*, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

2. GitHub – Hello World

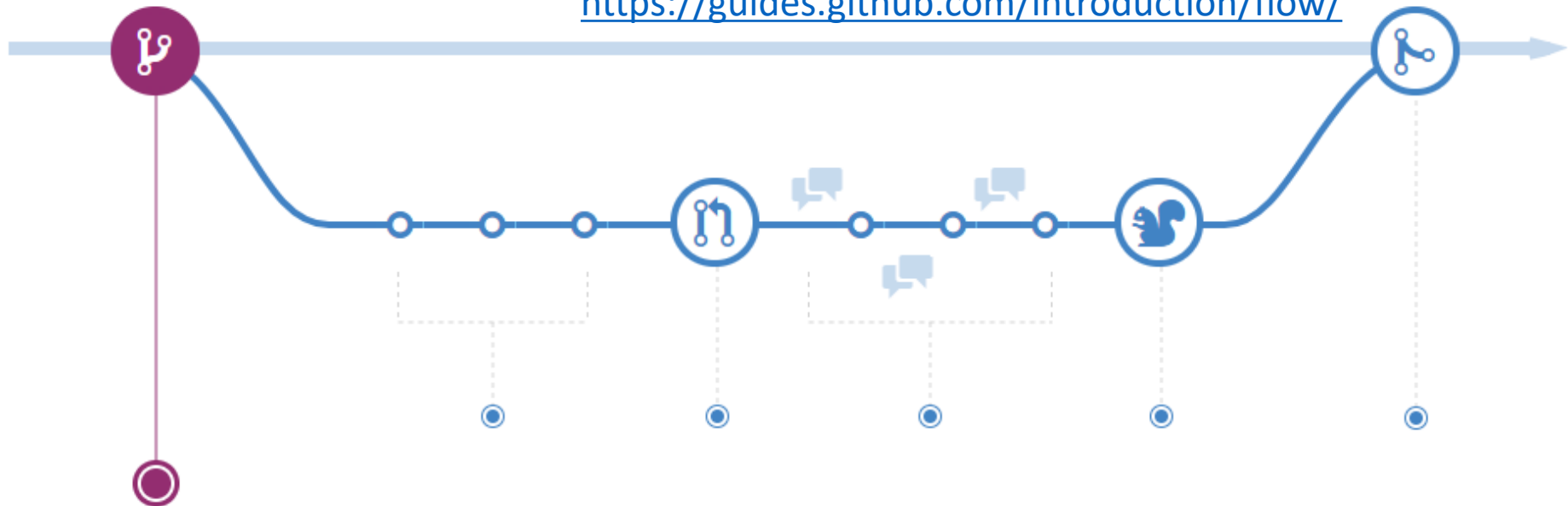
Step 5. Merge your Pull Request

In this final step, it's time to bring your changes together – merging your `readme-edits` branch into the `main` branch.

1. Click the green **Merge pull request** button to merge the changes into `main`.
2. Click **Confirm merge**.
3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.

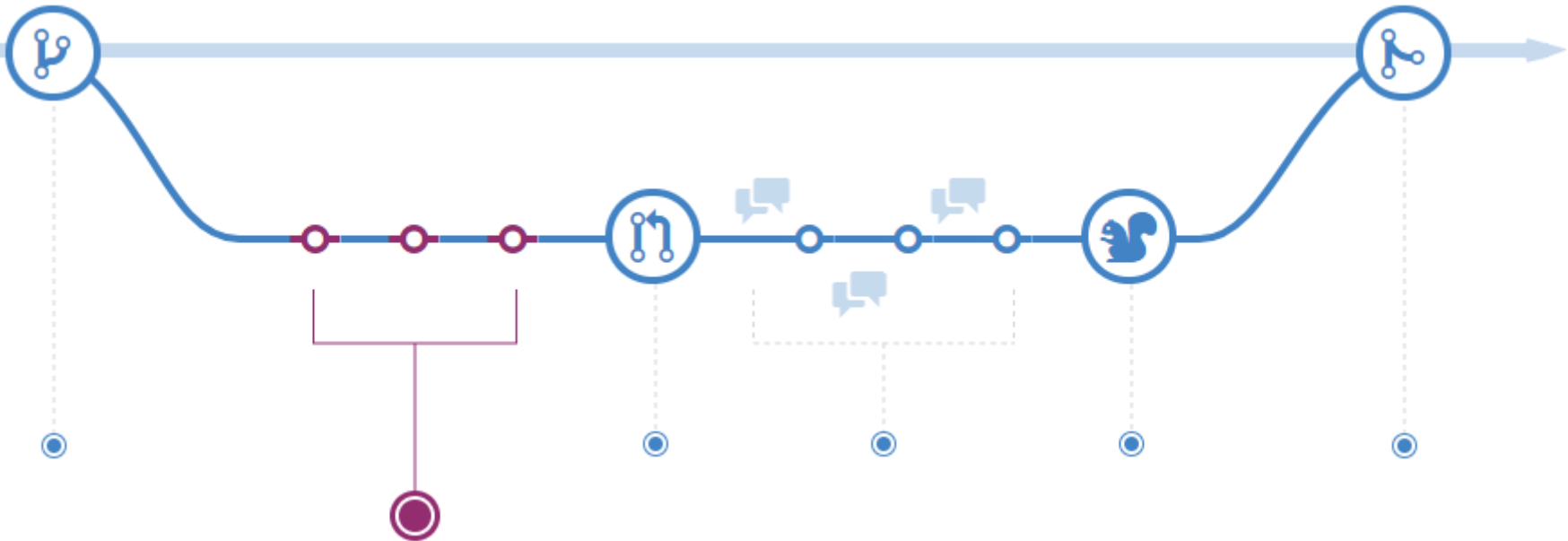
*The Flow: Creating a Branch

<https://guides.github.com/introduction/flow/>



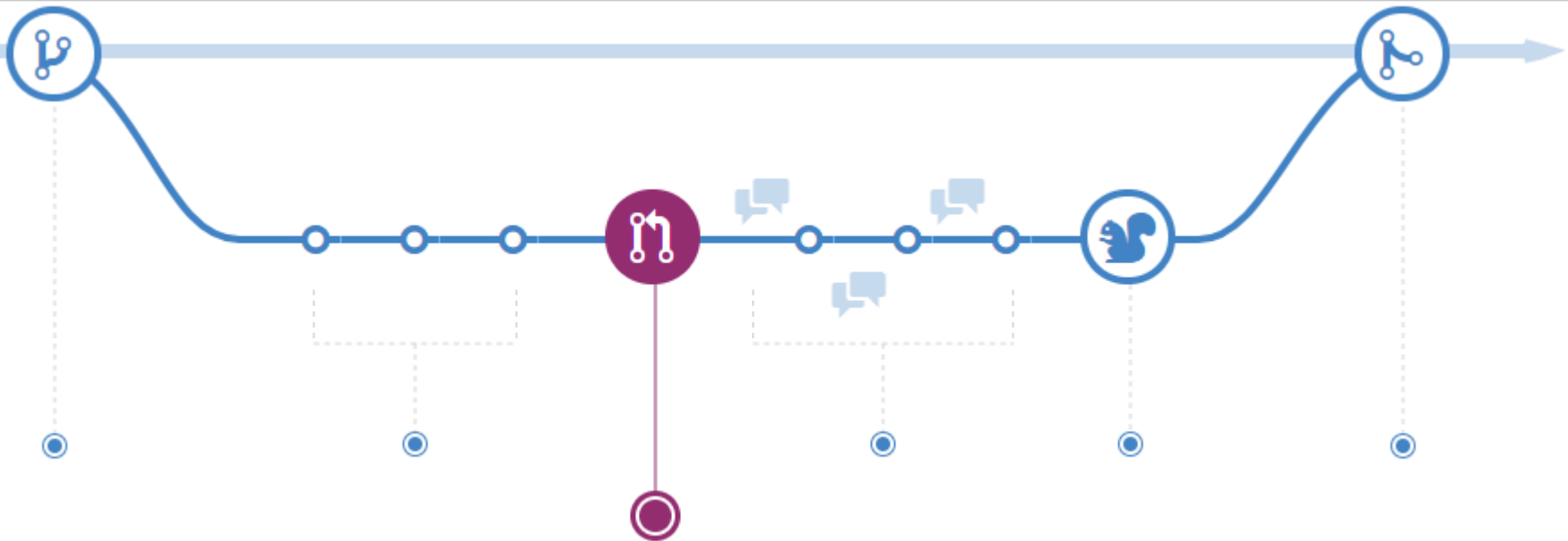
- When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. **Branching exists to help you manage this workflow.**
- **Changes you make on a branch don't affect the main branch**, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

*The Flow: Add Commits



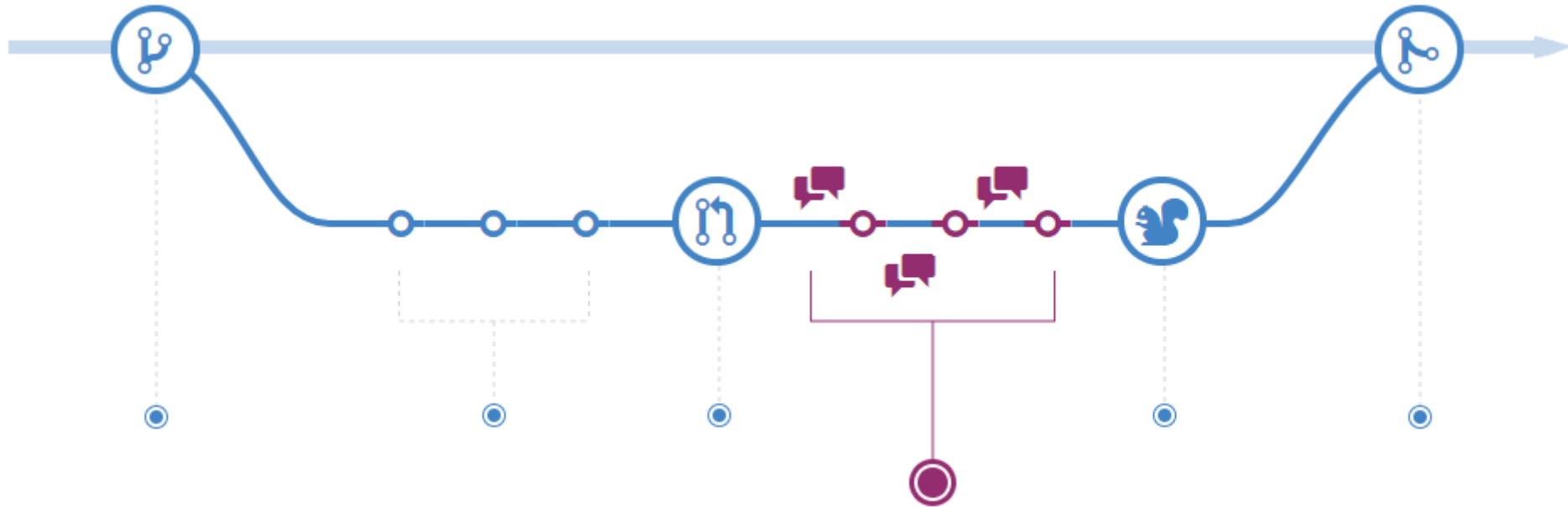
- Once your branch has been created, it's time to start making changes. **Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch.** This process of adding commits keeps track of your progress as you work on a feature branch.
- Commits also create a transparent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

*The Flow: Open a Pull Request



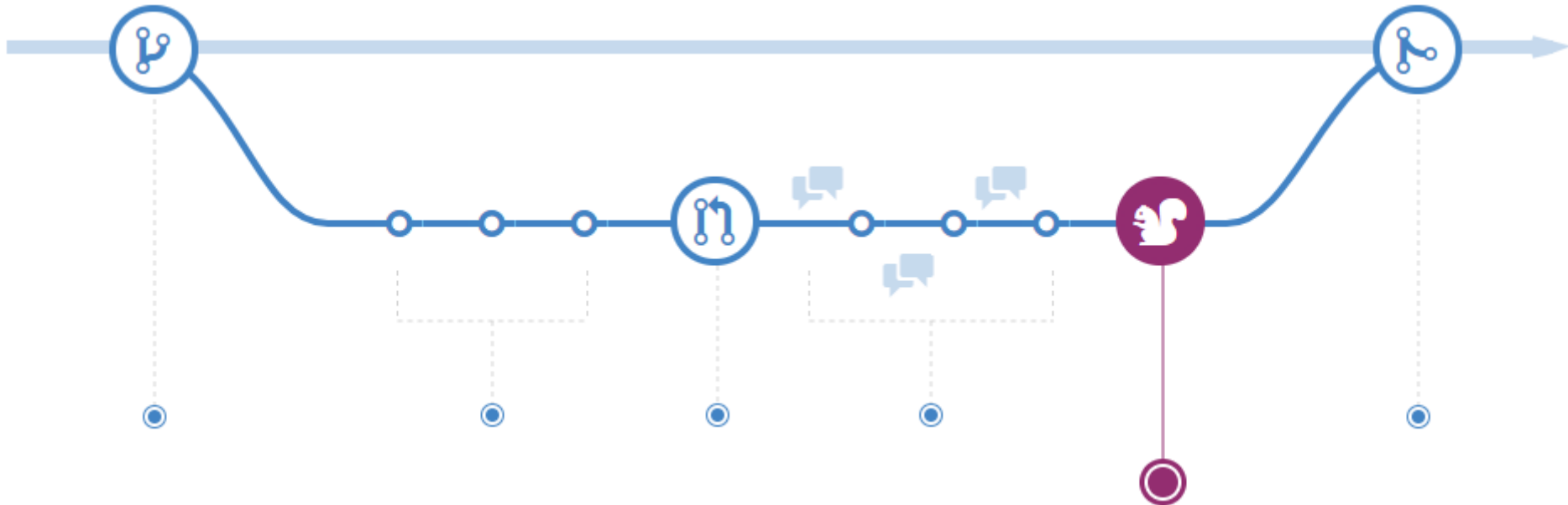
- **Pull Requests initiate discussion about your commits.** Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.
- **You can open a Pull Request at any point during the development process:** when you have 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.

*The Flow: Discuss and Review Your Code



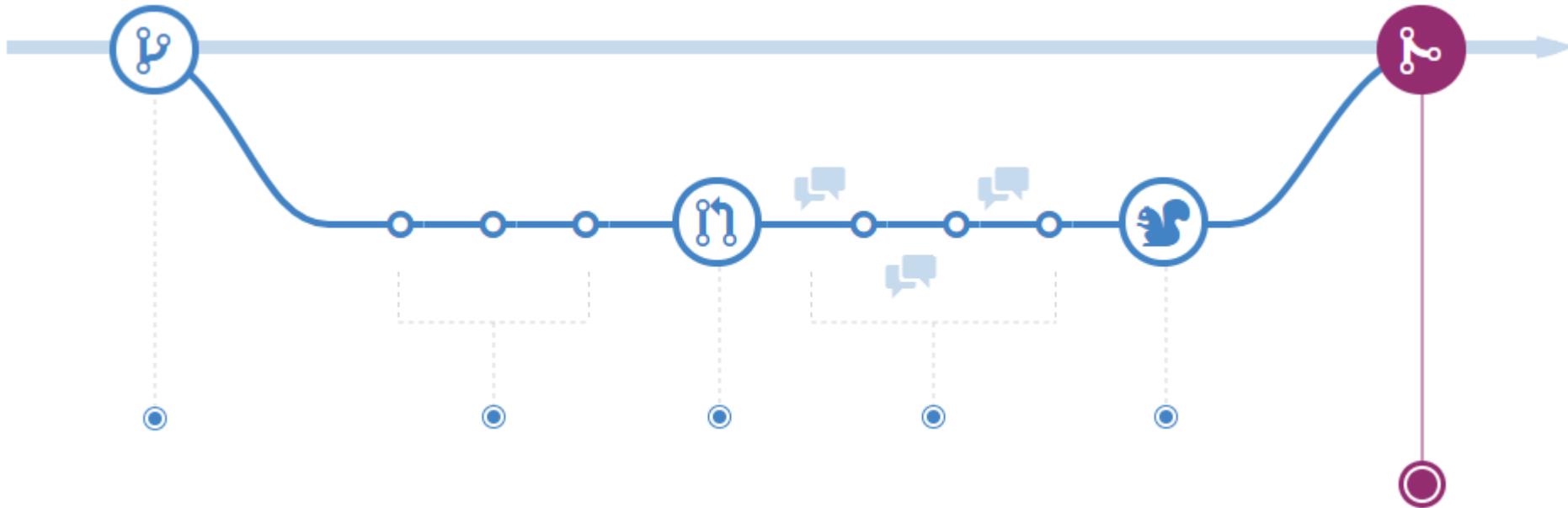
- **Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments.** Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage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

*The Flow: 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 main branch into production.

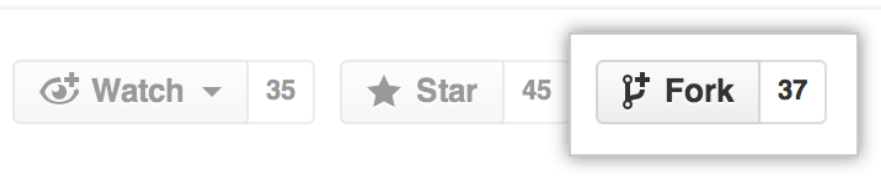
*The Flow: Merge



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

*Fork

- A fork is a copy of a repository.
- Forking a repository allows you to freely experiment with changes without affecting the original project.
- Most commonly, **forks** are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.



*Lab Exercise #2: Forking a project

- Find an open source project that you are interested in from GitHub
- Fork it (you must be logged in)
- * See the next slides for an example