Code Versioning System
Git and Github

Rony Setyawan, S.T., M.Kom.

Outline

👀 Git and GitHub Introduction
📚 Basic Git Concepts
📝 Working with Commits
🌐 Creating a Repository & Pushing Changes
🌳 Exploring Branches
💪 Exercise 2

👀 Git Definition – Git & GitHub Introduction

Git is a **distributed version control system (VCS)** that lets developers track changes to their code over time.

It allows multiple people to **work on the same project without overwriting one another's changes**. Git is designed to handle everything from small to very large projects quickly and efficiently.

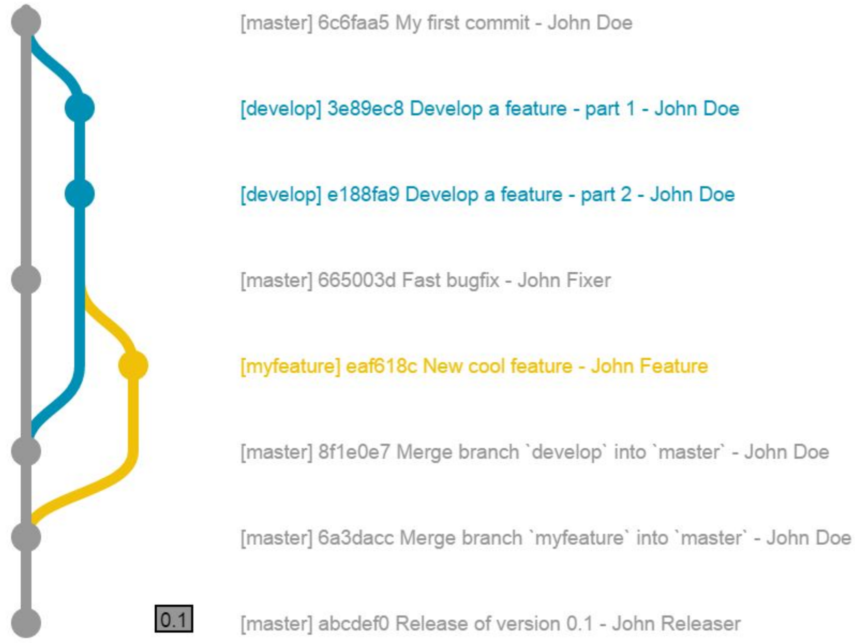
👀 GitHub Definition - Git & GitHub Introduction

In contrast, GitHub is a **web-based repository hosting service** for Git repositories.

Developers can keep their creations online with this platform, which facilitates easier collaboration with others. GitHub provides tools for project management and teamwork, such as pull requests, issue tracking, and project management software. For developers, it's essentially a social network where they can work together on projects, exchange code, and contribute to open-source projects.

👀 Why Git - Basic Git Concepts

The importance of Git and GitHub for developers lies in their ability to enhance productivity, collaboration, and code management.


[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe
[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe
0.1
[master] abcdef0 Release of version 0.1 - John Releaser

Think of Git like a tree. In this tree, there are many **branches** and each **(which represent commits)**.

Keywords:
Repository, Commit, Branch, and Merge

Visualize Your Git Repositories

5 Super Easy-to-use Tools

📚 Repository - Basic Git Concepts

A repository—often abbreviated as "repo"—is where your **project is kept in storage**.

It's like a big folder on your computer that
keeps track of all the files and changes made to those files over time.

📚 Commits – Basic Git Concepts

A commit is like a
snapshot of your project at a specific point in time.

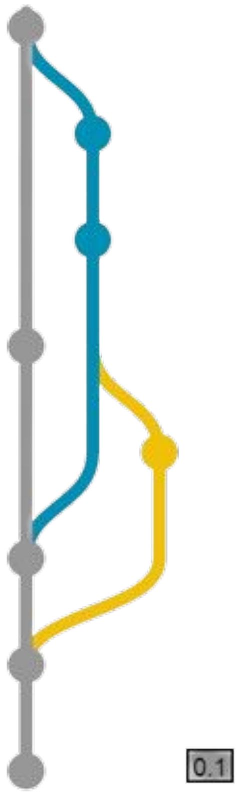
When you make changes to your project, **you can save these changes as a commit**. This way, you can go back to any point in your project's history and see exactly what the project looked like at that time.

📚 Branch - Basic Git Concepts

This branch is like a **copy of your project that you can work on without affecting the main project**.

Once you're happy with your changes, you can **merge them back** into the main project.

📚 Branches – Basic Git Concepts



[master] 6c6faa5 My first commit - John Doe

[develop] 3e89ec8 Develop a feature - part 1 - John Doe

[develop] e188fa9 Develop a feature - part 2 - John Doe

[master] 665003d Fast bugfix - John Fixer

[myfeature] eaf618c New cool feature - John Feature

[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe

[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe

[master] abcdef0 Release of version 0.1 - John Releaser

📚 Merge – Basic Git Concepts

Merging is the process of **taking the changes you made in a branch and combining them** with the main project.

Git will try to combine the changes automatically, but if there are any conflicts, you'll need to resolve these conflicts manually.

👀 Question – Git & GitHub Introduction

What do software engineers do in the event of a building fire?

👀 Question – Git & GitHub Introduction

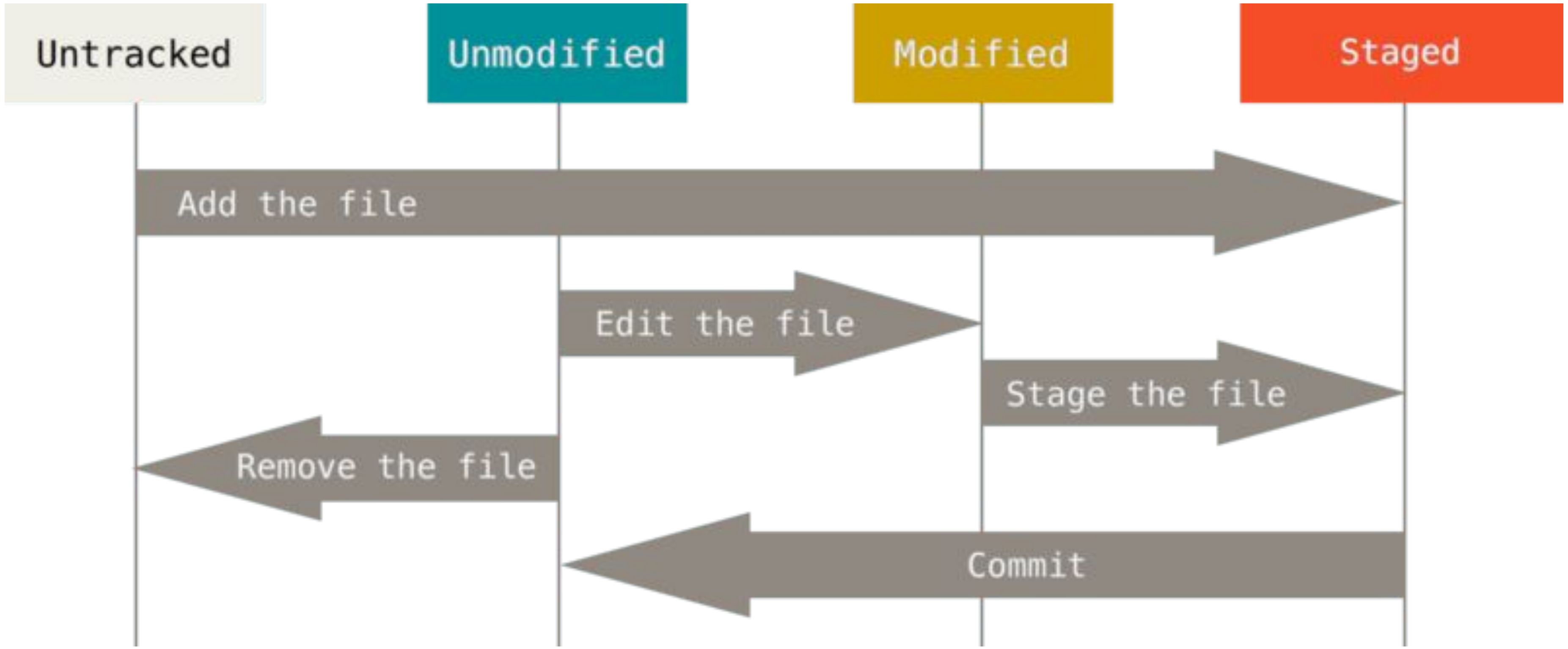
📝 Status Lifecycle – Working with Commits

each file in your working directory can be in one of two states: **tracked or untracked**

Tracked files are files that Git knows about.

Untracked files are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area.

📝 Status Lifecycle – Working with Commits



📝 Understanding Commits – Working with Commits

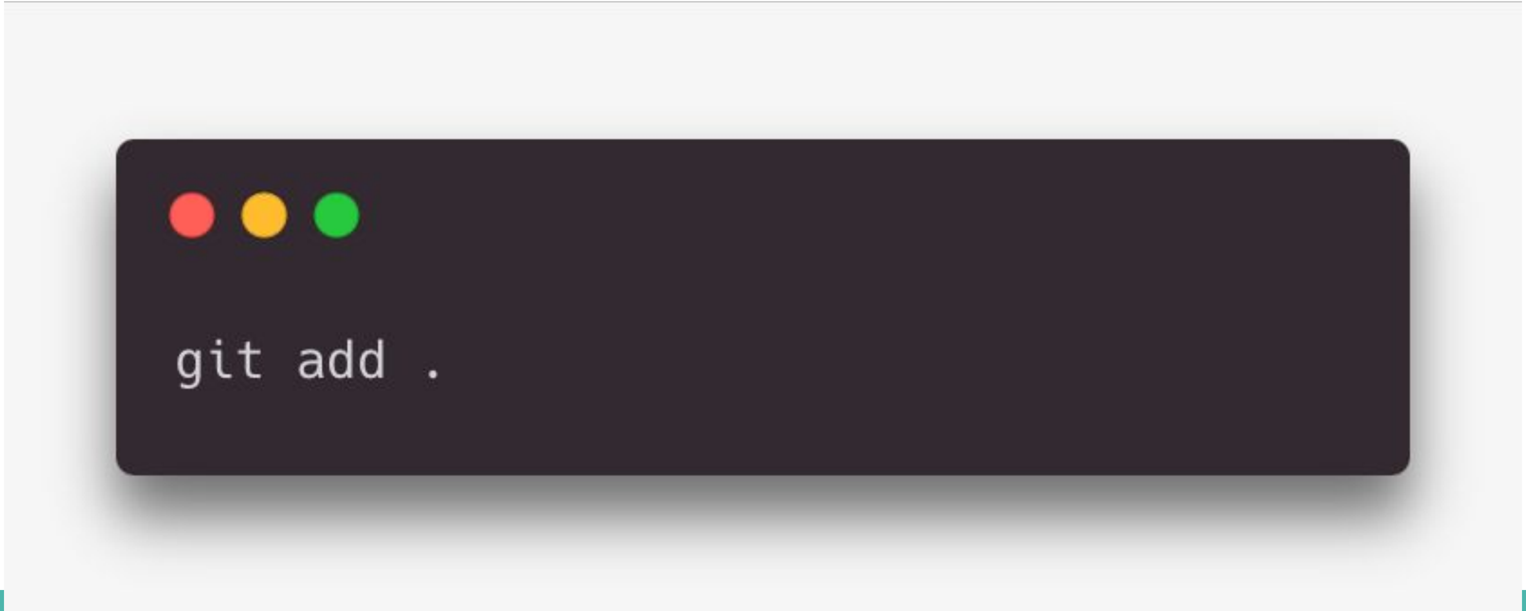
A repository need to be Initiated by running **<u>git init</u>** command

Before initializing a repository, make sure you are **already on your working directory (folder)**

```
git init
```

📝 Understanding Commits – Working with Commits

Make some changes on your project, and stage the files using **git add .** which will **add all files into staged** phase



```
git add .
```

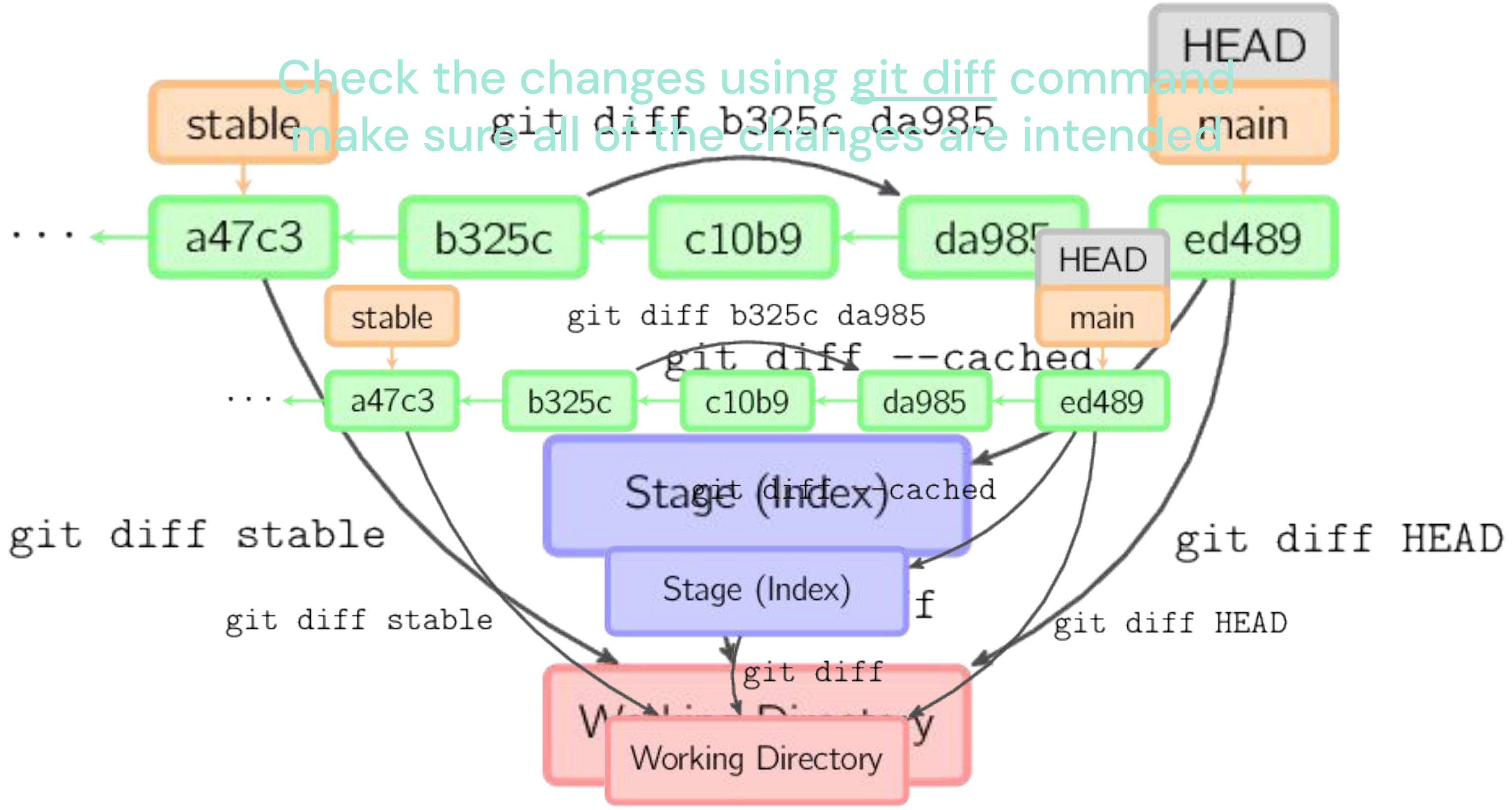
📝 Understanding Commits – Working with Commits

Check the **paths that have differences** between the index file and the current HEAD commit by using **`git status`**

```
git status
```

📝 Understanding Commits – Working with Commits

Check the changes using git diff command
make sure all of the changes are intended

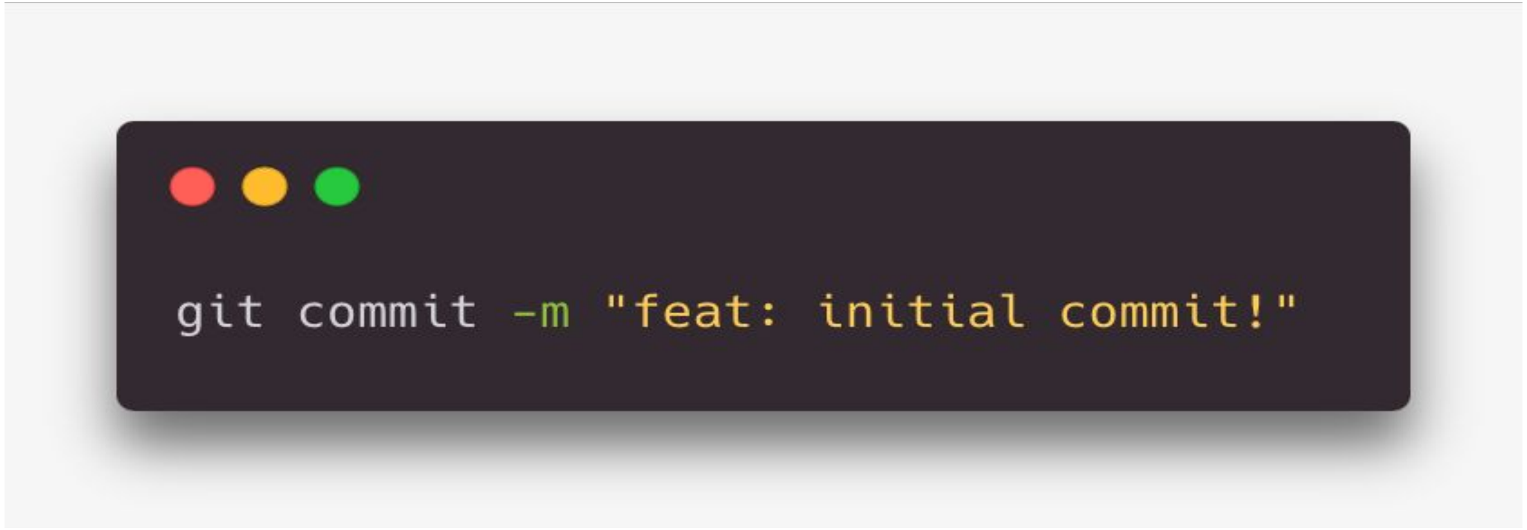


📝 Understanding Commits - Working with Commits

Commit all of the staged files by using this command

git commit -m <your message>

You can follow the **Conventional Commits** convention for the commit message



```
git commit -m "feat: initial commit!"
```

📝 Understanding Commits – Working with Commits

`git add .`	To begin tracking the **ALL** file
`git status`	To Displays paths that have **differences between the index file and the current HEAD commit**
`git commit -m`	To make a commit **with message**
`git diff`	To check **changes**

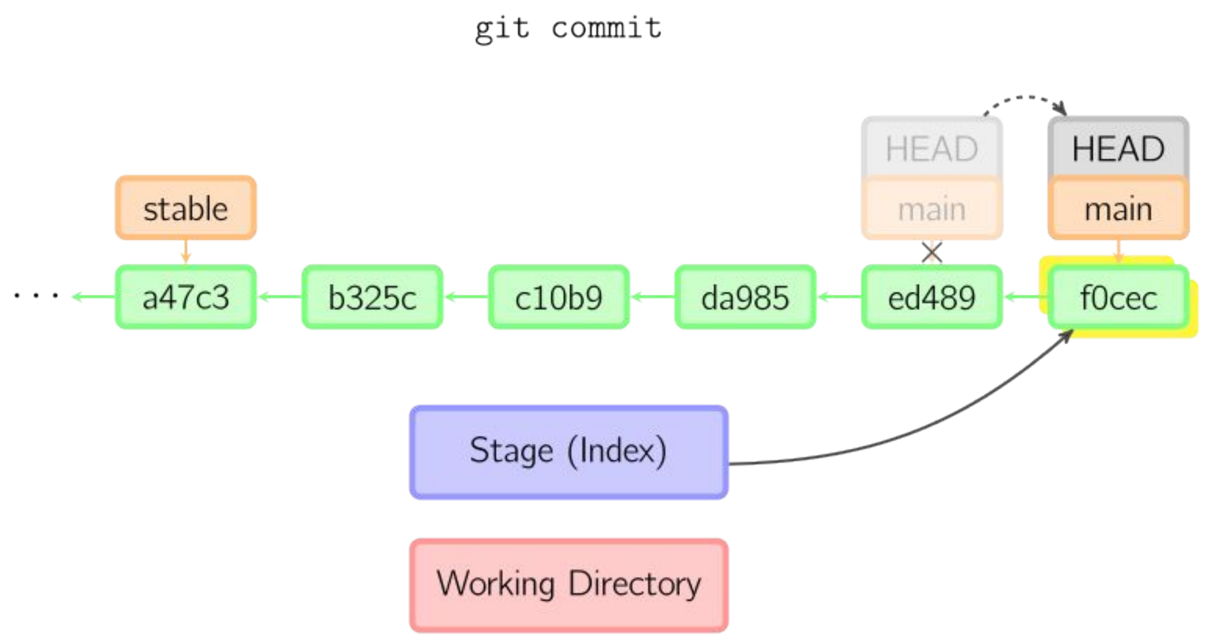
https://git-scm.com/docs/git-add
https://git-scm.com/docs/git-status
https://git-scm.com/docs/git-commit
https://git-scm.com/docs/git-diff

📝 Understanding Commits – Working with Commits

Congrats! You made a
snapshot of your project at a specific point in time! 😉

📝 Understanding Commits – Working with Commits

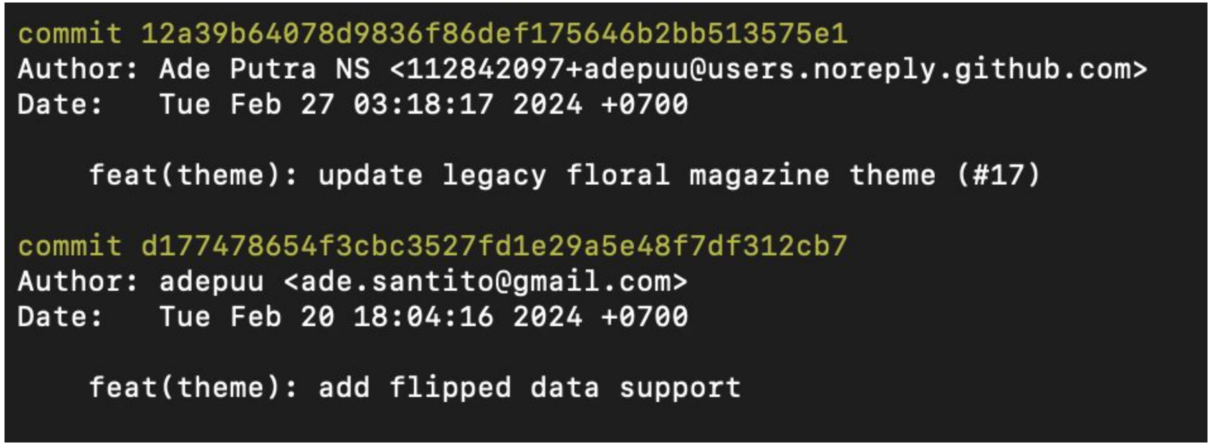
Behind the scene



📝 Commit History – Working with Commits

Each **commit** in Git is **identified by a unique SHA-1 hash** (a 40-character string of hex digits)

This hash allows Git to reference commits, making it easier to track changes and revert to previous versions



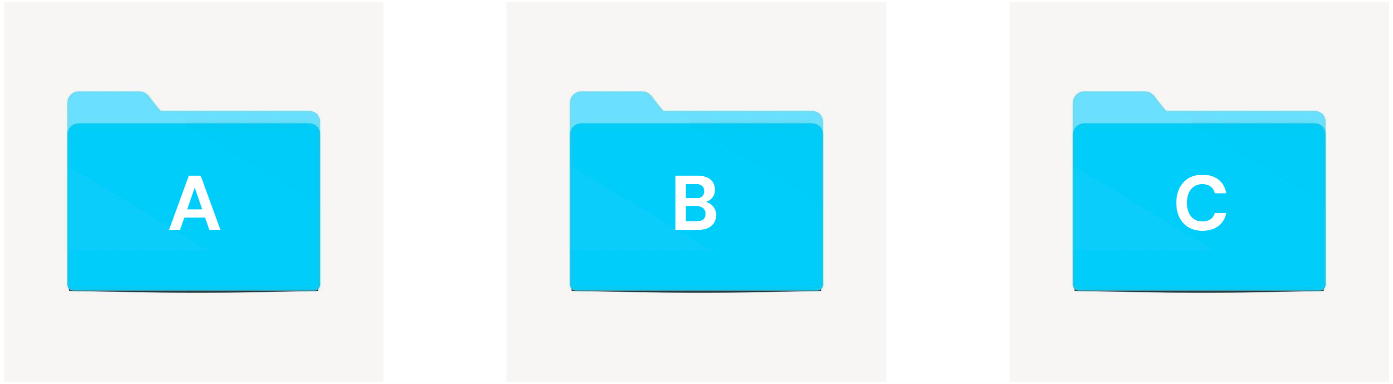
```
commit 12a39b64078d9836f86def175646b2bb513575e1
Author: Ade Putra NS <112842097+adepuu@users.noreply.github.com>
Date:   Tue Feb 27 03:18:17 2024 +0700

    feat(theme): update legacy floral magazine theme (#17)

commit d177478654f3cbc3527fd1e29a5e48f7df312cb7
Author: adepuu <ade.santito@gmail.com>
Date:   Tue Feb 20 18:04:16 2024 +0700

    feat(theme): add flipped data support
```

📝 The Perfect Commit – Working with Commits

1. Add the **right** changes
2. Compose a **good** commit message

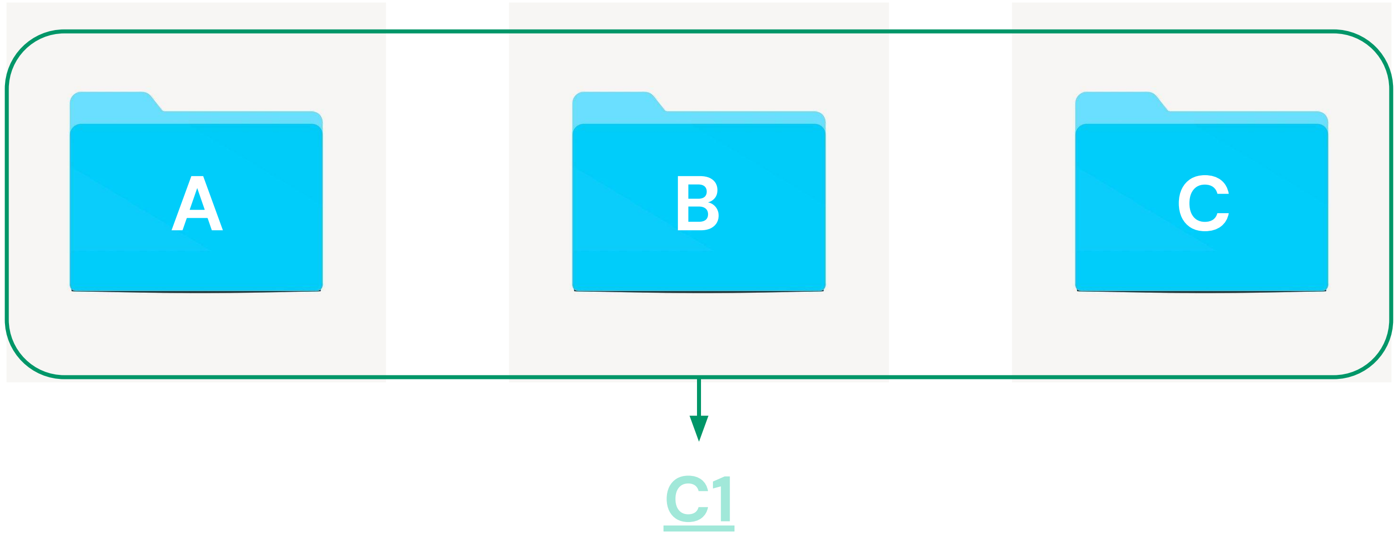
📝 Add The Right Changes – Working with Commits

The Perfect Commit



📝 Add The Right Changes – Working with Commits

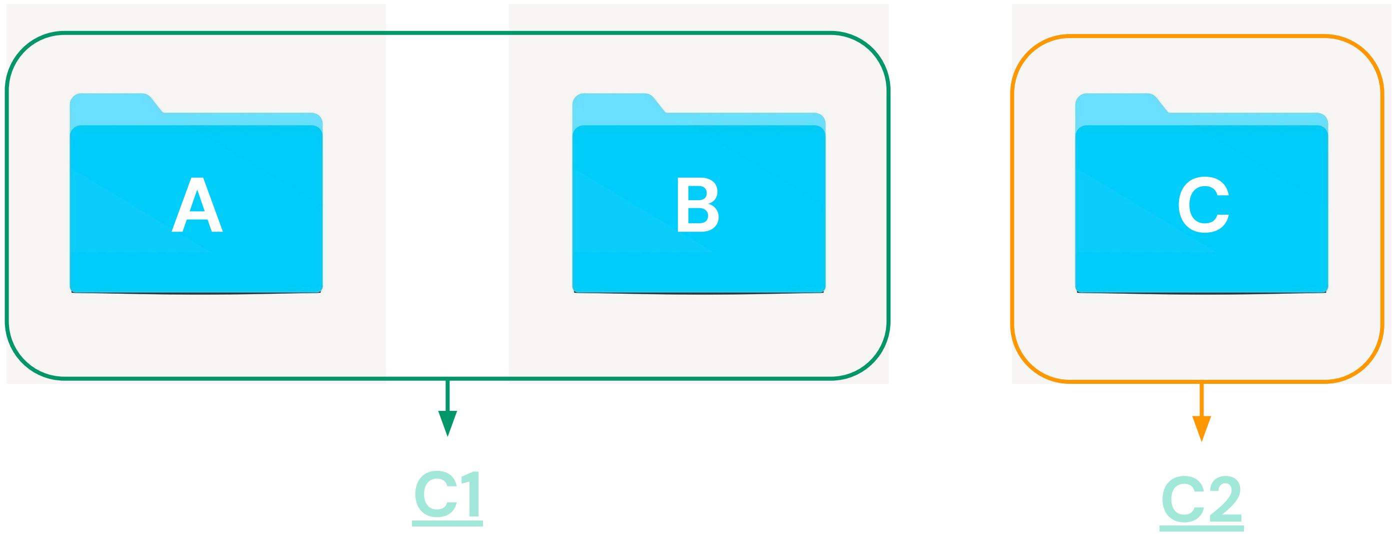
The ~~Perfect~~ Commit



📝 Add The Right Changes – Working with Commits

The ~~Perfect~~ Better Commit

Separate your topics!



📝 Good Commit Message– Working with Commits

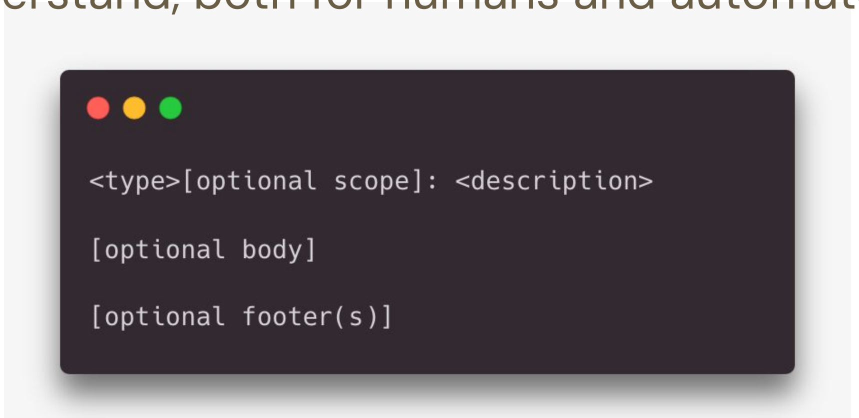
A good commit message should be **clear, concise, and provides a meaningful description of the changes made in the commit**.

It should effectively communicate the intent of the changes to other developers, stakeholders, and even to the future self.

📝 Conventional Commit – Working with Commits

Angular-style commit messages follow a structured format that **includes a header, body, and an optional footer**.

This format is designed to make commit histories easier to read and understand, both for humans and automated tools



```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

https://www.conventionalcommits.org/en/v1.0.0/#summary

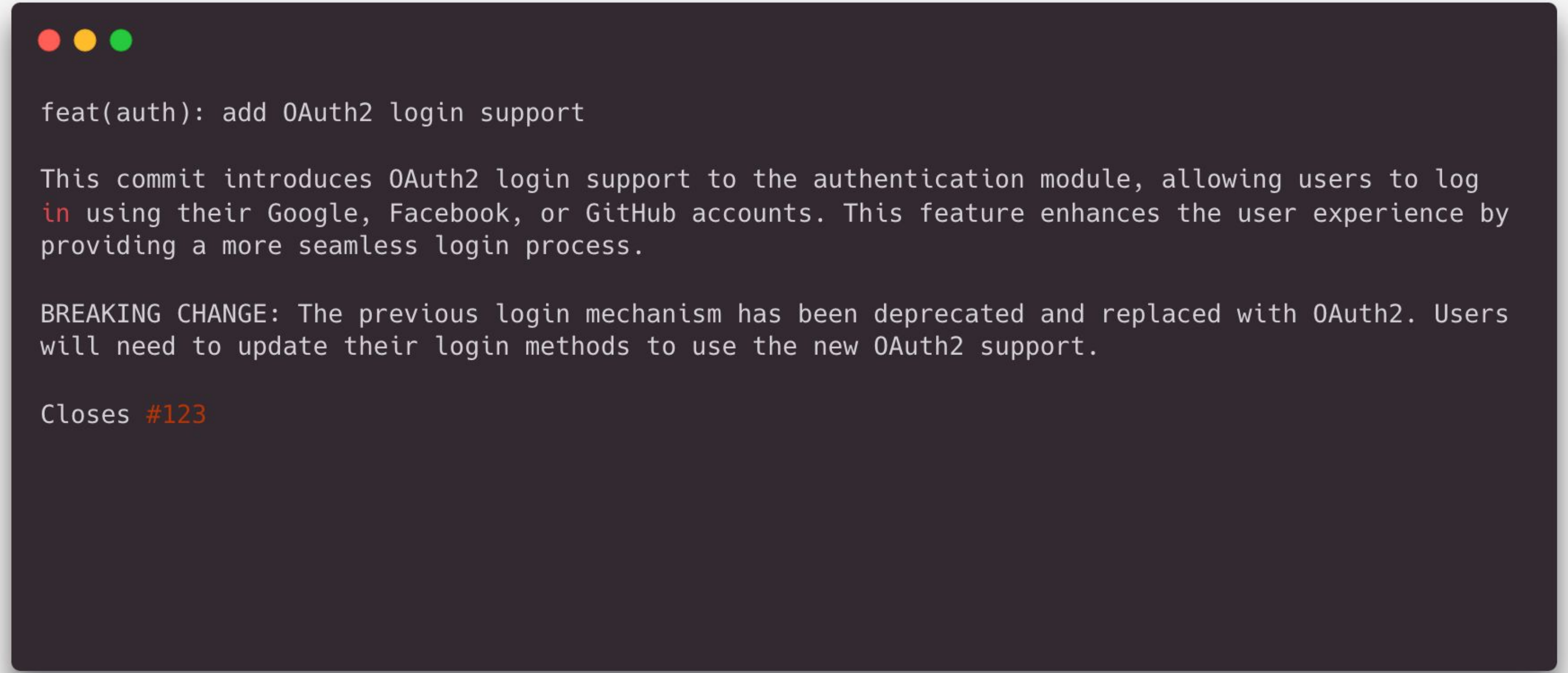
📝 Conventional Commit – Working with Commits

Type list :

● **build**: Build related changes (eg: npm related/ adding external dependencies)
● **chore**: A code change that external user won't see (eg: change to .gitignore file or .prettierrc file)
● **feat**: A new feature
● **fix**: A bug fix
● **docs**: Documentation related changes
● **refactor**: A code that neither fix bug nor adds a feature. (eg: You can use this when there is semantic changes like renaming a variable/ function name)
● **perf**: A code that improves performance
● **style**: A code that is related to styling
● **test**: Adding new test or making changes to existing test

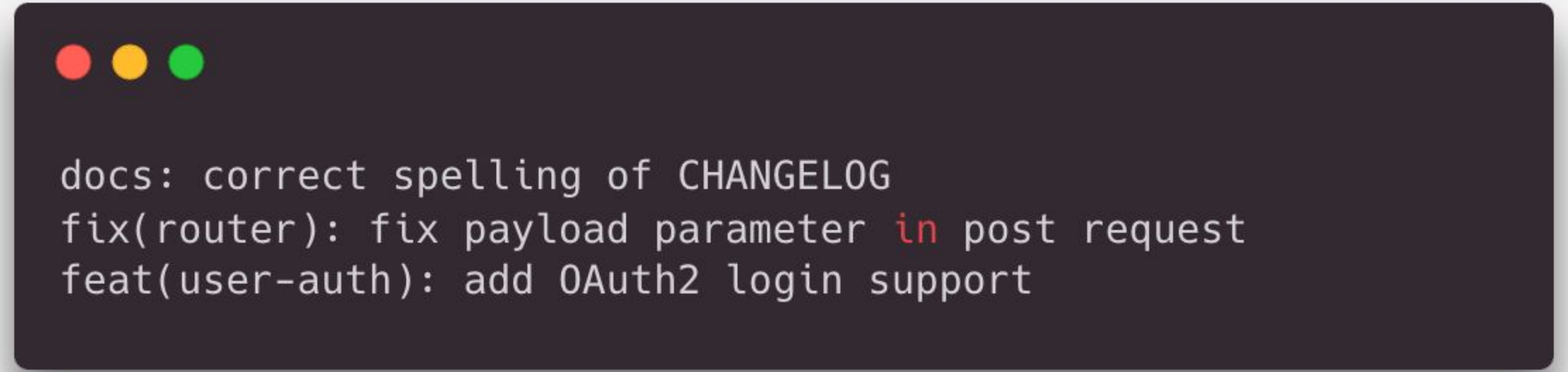
https://www.conventionalcommits.org/en/v1.0.0/#summary

📝 Example – Working with Commits



```
feat(auth): add OAuth2 login support

This commit introduces OAuth2 login support to the authentication module, allowing users to log
in using their Google, Facebook, or GitHub accounts. This feature enhances the user experience by
providing a more seamless login process.

BREAKING CHANGE: The previous login mechanism has been deprecated and replaced with OAuth2. Users
will need to update their login methods to use the new OAuth2 support.

Closes #123
```

📝 Example – Working with Commits



```
docs: correct spelling of CHANGELOG
fix(router): fix payload parameter in post request
feat(user-auth): add OAuth2 login support
```

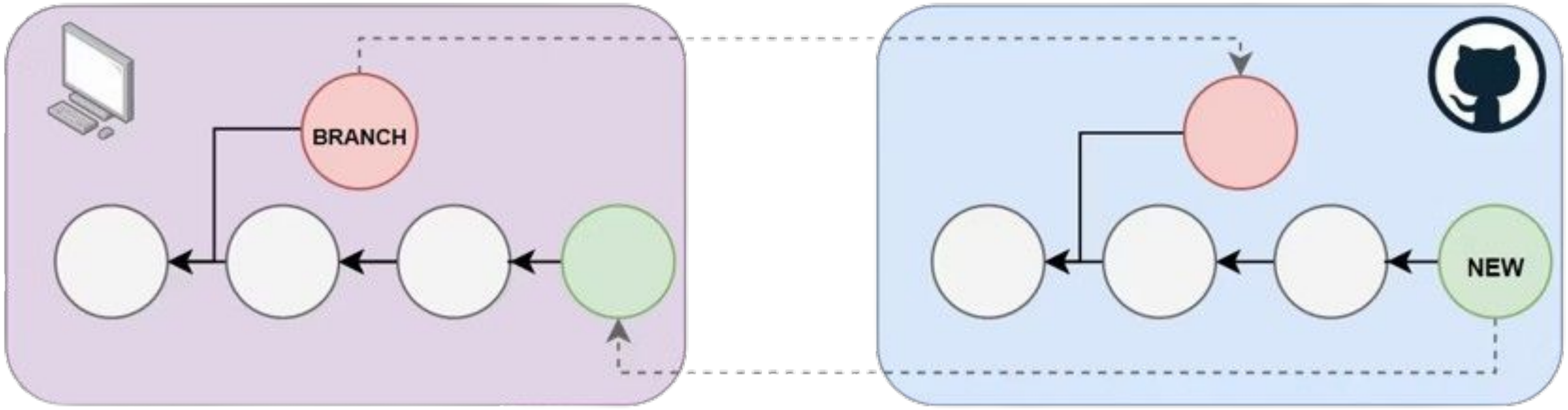
📝 The Bad & The Good – Working with Commits

Bad	Good
`update`	`docs: correct spelling of CHANGELOG`
`fix bugs`	`fix(router): fix payload parameter in post request`
`fix typo in readme`	`feat(user-auth): add OAuth2 login support`

🌐 Remote Repository – Creating a Repository & Pushing Changes

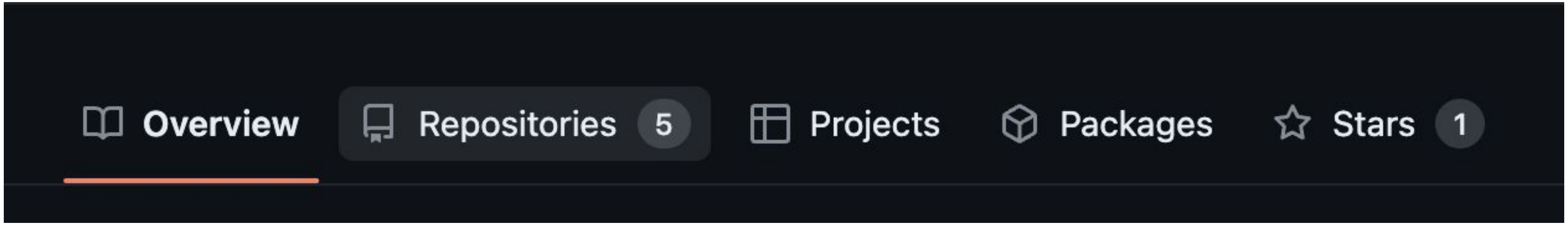
To collaborate using Git, we can create a remote repository that will be hosted on GitHub.

Local Repo **Remote Repo**

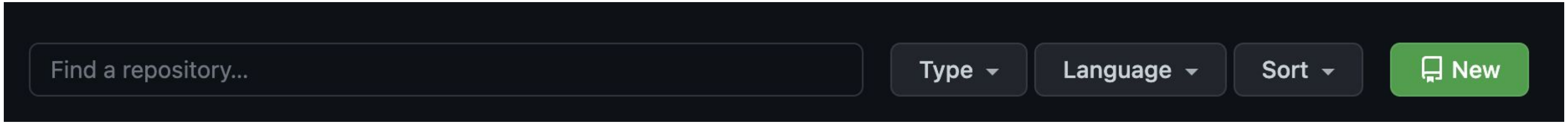


🌐 Create GitHub Repo – Creating a Repository & Pushing Changes

Step 1 – Go to “Repositories” tab on your GitHub profile page



Step 2 – Click “New” button and go to the next page



🌐 Create GitHub Repo – Creating a Repository & Pushing Changes

Step 3 – Fill the required fields (empty repo)

Give your repo a name

Leave the description empty, or fill if you have any

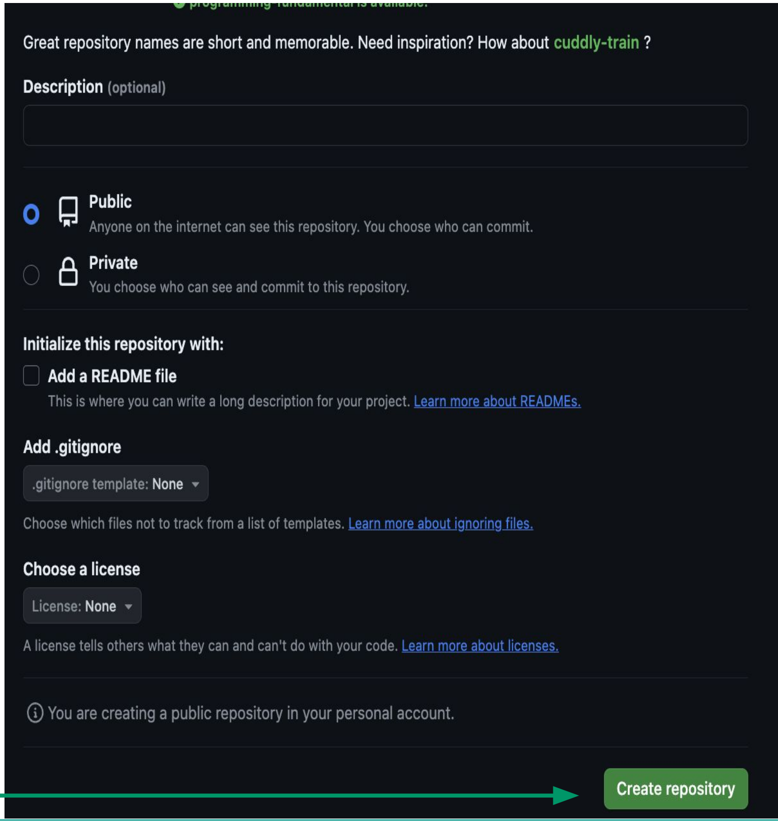
Make it a public repo

Uncheck the Add README file

Leave it to None

Leave it to None

Last click create



Great repository names are short and memorable. Need inspiration? How about cuddly-train ?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

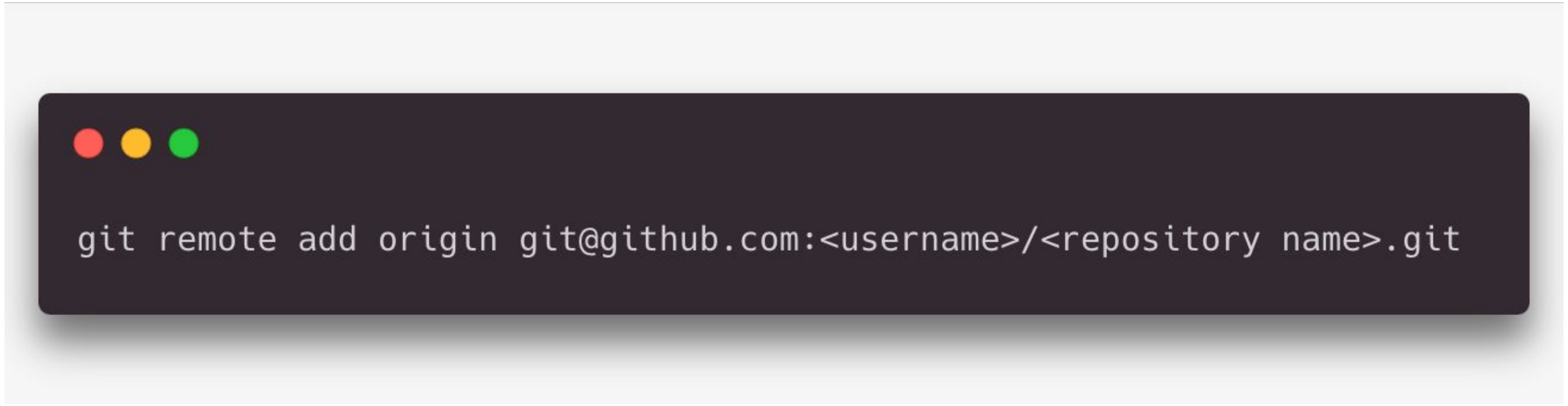
License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

You are creating a public repository in your personal account.

🌐 Create GitHub Repo – Creating a Repository & Pushing Changes

Step 4 – Add remote origin to the existing local Git Repo

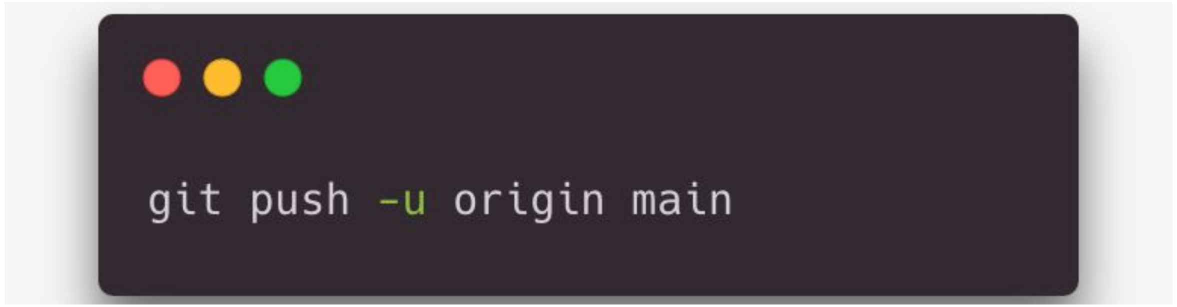


```
git remote add origin git@github.com:<username>/<repository name>.git
```

<username> should be replaced with your GitHub Username
<repository name> should be replaced with the Repository Name we just created

🌐 Create GitHub Repo – Creating a Repository & Pushing Changes

Step 5 – Push the local changes to GitHub



```
git push -u origin main
```

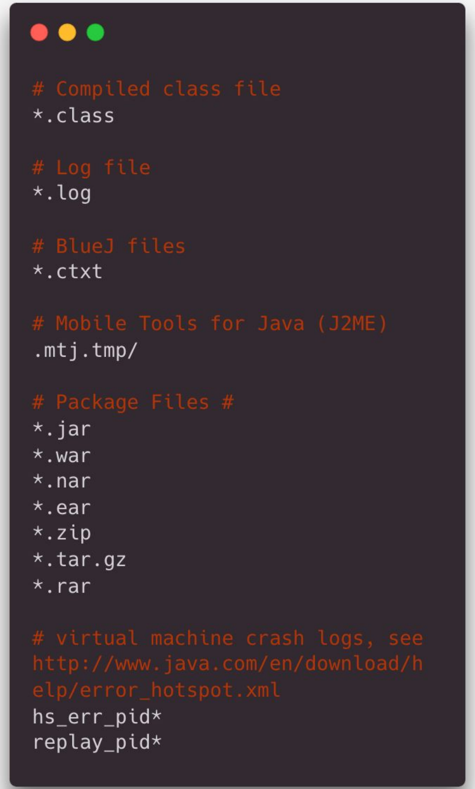
git push origin <branch>

Only use -u flag if we **haven't set the remote upstream**

🌐 Ignoring Files – Creating a Repository & Pushing Changes

Adding a **.gitignore** file to every Git repository, particularly Java projects, is critical for efficiently managing the repository's contents,

exchanging ignore rules with team members, and maintaining **a clean and focused project structure**.



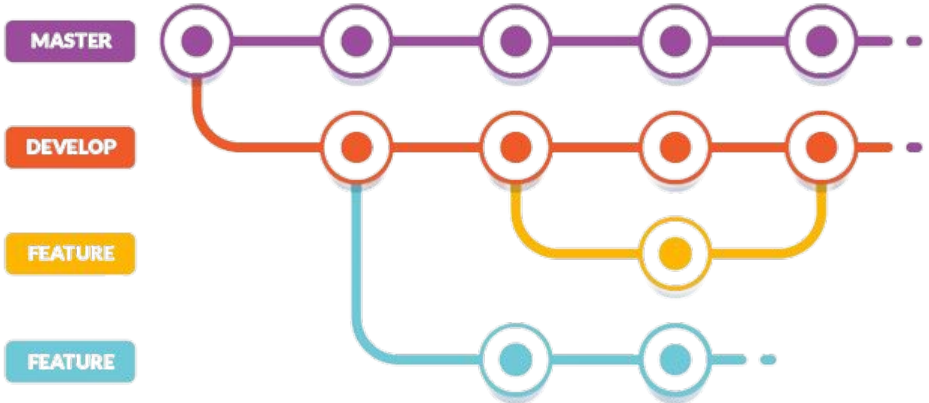
```
# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see
http://www.java.com/en/download/h
elp/error_hotspot.xml
hs_err_pid*
replay_pid*
```

https://github.com/github/gitignore

🌳 Introduction to Branches – Exploring Branches

Git branches are essentially
independent streams of development.

They enable developers to work on various features or fixes **without disrupting the main codebase**.



🌳 Introduction to Branches – Exploring Branches

A written convention

Agree on a branching Workflow on your team

1. Git allows you to create branches, but **it doesn't tell you how to use them**
2. You need a **written best practices** ideally on your team to avoid mistakes and collision
3. It highly **depends on your team / team size** on your project on how you handle release
4. It helps to onboard new team members "**this is how we work here**"

🌳 Branching Strategies – Exploring Branches

A branching strategy is a **set of rules** that software development teams adopt for **writing, merging, and deploying code** using a version control system like Git.

It outlines how developers should interact with a shared codebase

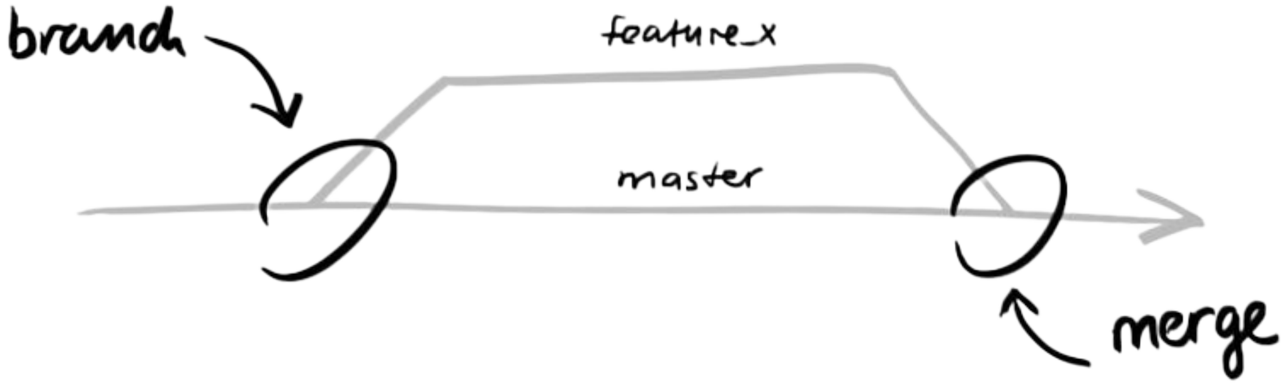
1
GitHub Flow

2
GitFlow

3
GitLab Flow

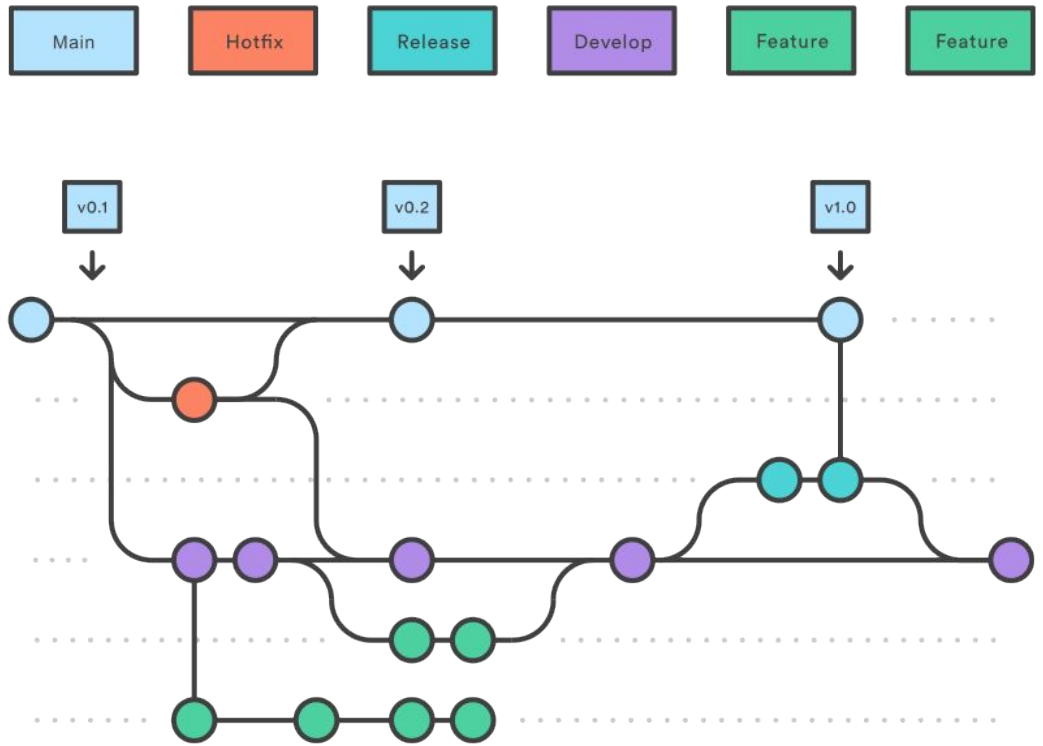
🌳 GitHub Flow Strategy – Exploring Branches

GitHub Flow is best suited for **smaller teams or projects** that do not require **handling many code versions**. It focuses on having a **single main branch** with production-ready code



🌳 GitFlow Strategy – Exploring Branches

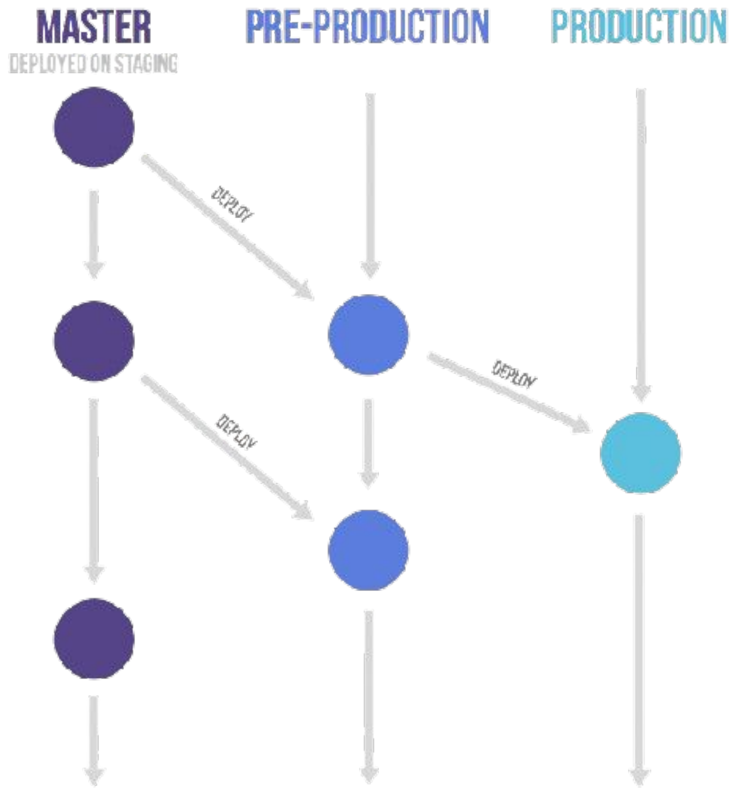
GitFlow is a more complex branching technique that uses a **variety of branches** for diverse purposes, including **feature branches, release branches, and hotfix branches**.



🌳 GitLab Flow Strategy - Exploring Branches

GitLab Flow is a more straightforward alternative to GitFlow that combines **feature-driven development and issue tracking**.

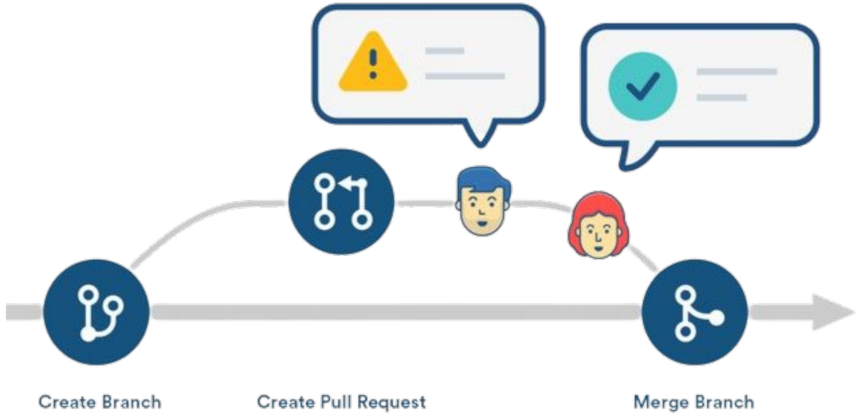
It immediately integrates with the main branch and is ideal for managing several environments, such as a staging environment



🌳 Pull Request – Exploring Branches

A pull request in Git allows a **developer to suggest modifications to a project**.

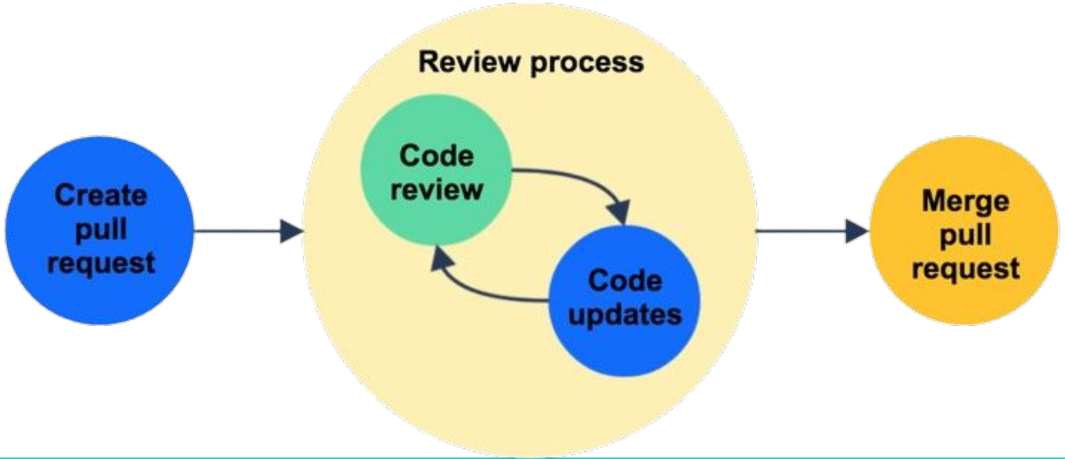
It is a formal request for another developer (often the project maintainer) to pull changes from one branch of a repository into another, usually from a feature branch to the main branch.



🌳 PR Review – Exploring Branches

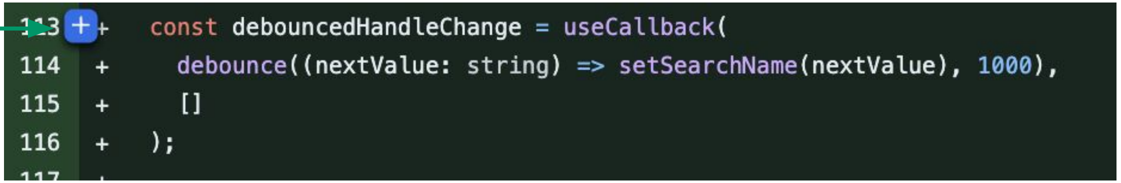
Pull request code review is the process of **assessing changes** suggested by other developers in a pull request (PR) before integrated into the main source.

This approach is **critical** for discovering possible problems, such as defects, performance concerns, or deviations from coding standards, as well as encouraging team engagement and information sharing.

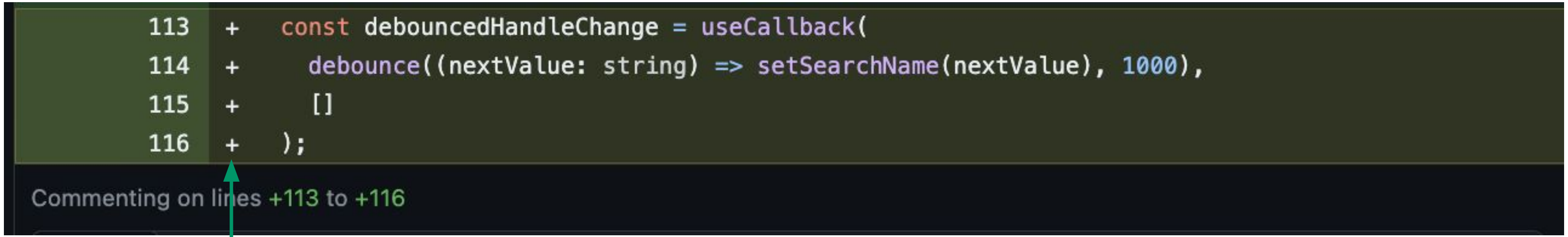


🌳 PR Review – Exploring Branches

Single line comment



```
113 + const debouncedHandleChange = useCallback(
114 +   debounce((nextValue: string) => setSearchName(nextValue), 1000),
115 +   []
116 + );
```



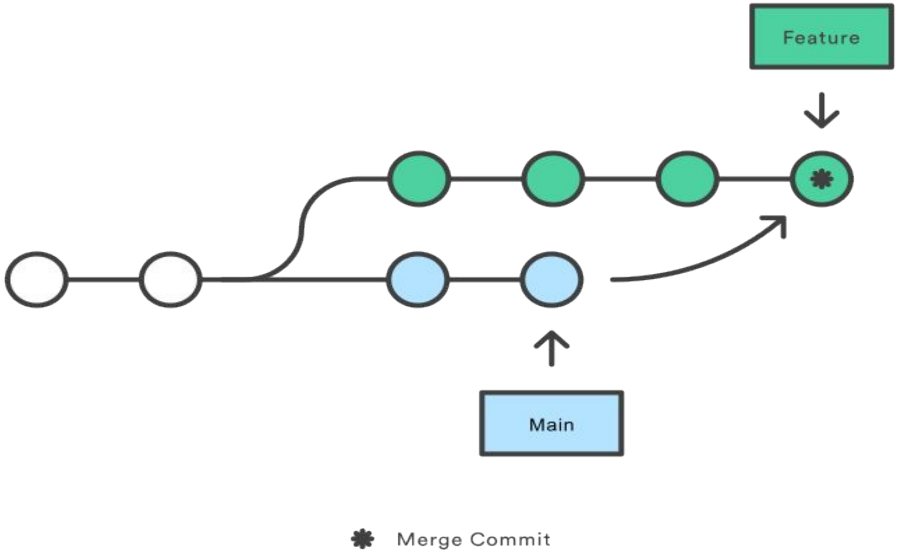
```
113 + const debouncedHandleChange = useCallback(
114 +   debounce((nextValue: string) => setSearchName(nextValue), 1000),
115 +   []
116 + );
```

Commenting on lines +113 to +116

Drag the + icon for Multi line comment

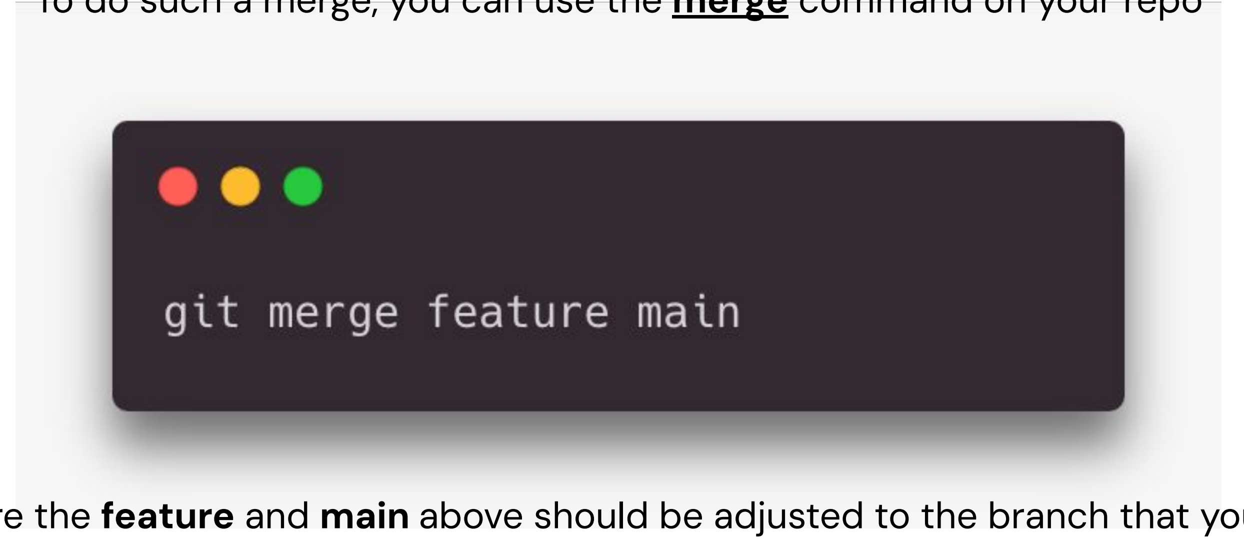
🌳 Merge – Exploring Branches

A merge creates a **new commit** that incorporates changes from other commits that will creates a new **“merge commit”** in the **feature branch** that ties together the histories of both branches, giving you a branch structure that looks like this:



🌳 Merge – Exploring Branches

To do such a merge, you can use the **merge** command on your repo

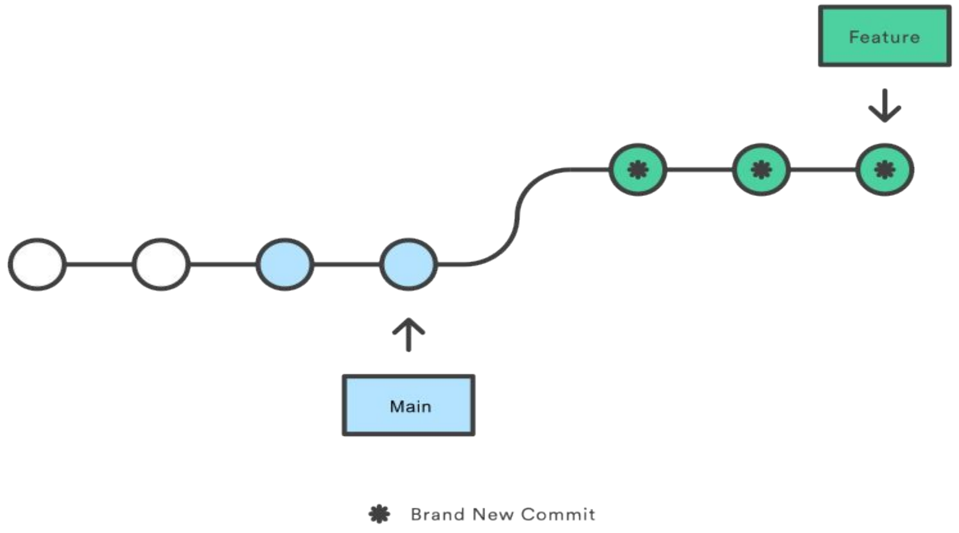


```
git merge feature main
```

Where the **feature** and **main** above should be adjusted to the branch that you are going to merge

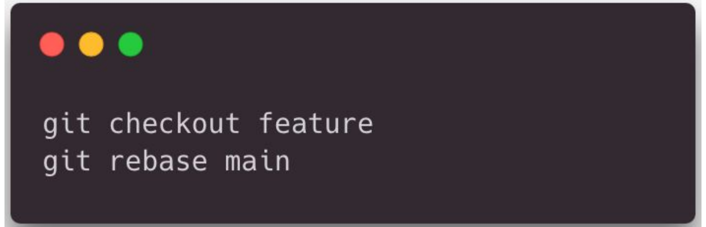
🌳 Rebase – Exploring Branches

Rebase **moves the entire feature branch to begin on the tip of the main branch**, effectively incorporating all of the new commits in main. Instead of using a merge commit, rebasing re-writes the project history by creating brand new commits for each commit in the original branch.



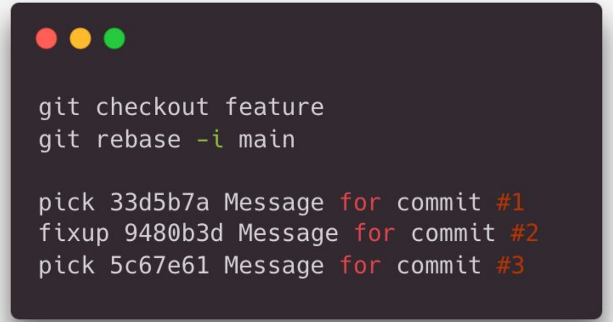
🌳 Rebase – Exploring Branches

To do rebasing, you can use the **rebase** command on your repo



```
git checkout feature
git rebase main
```

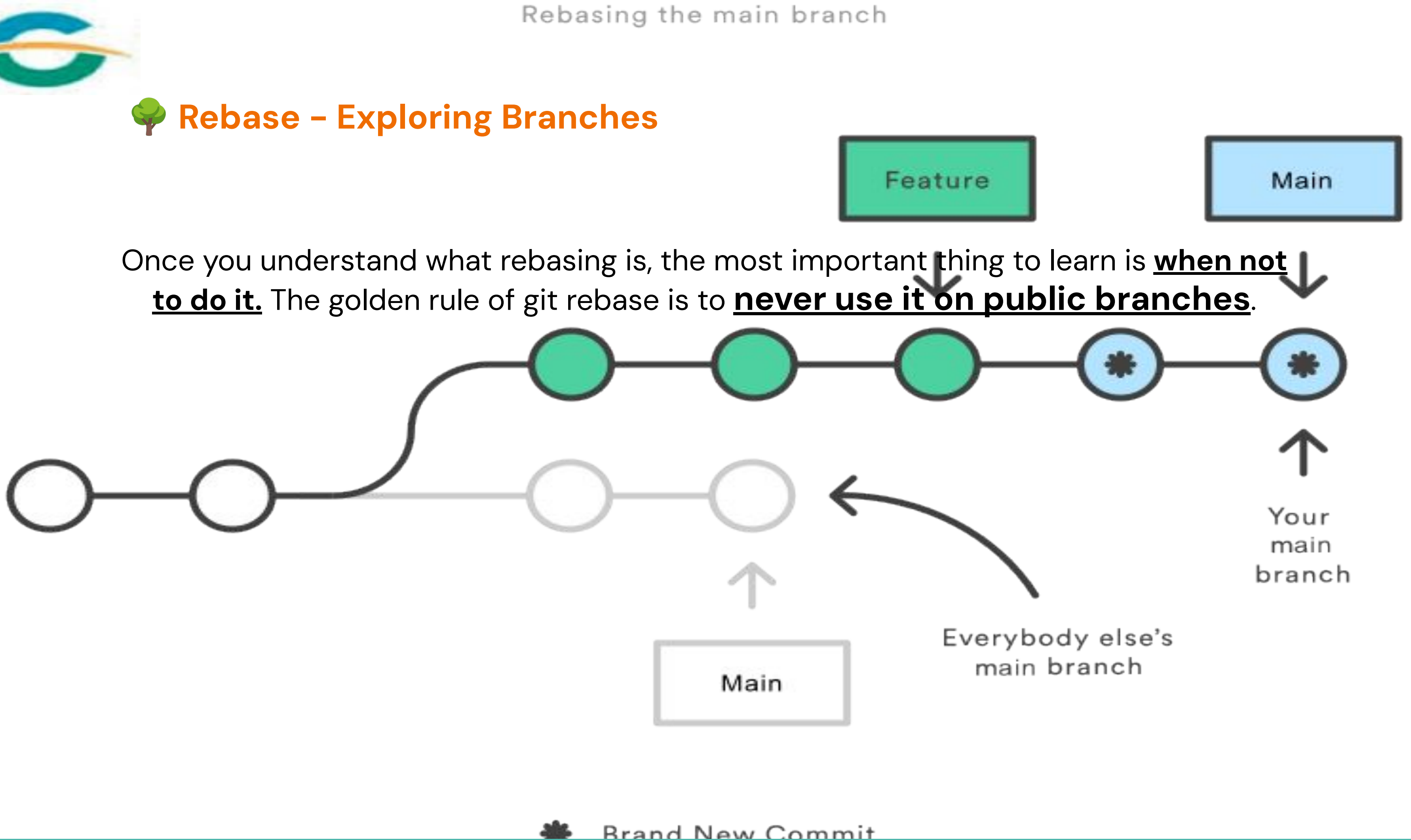
There is also interactive rebasing to alter commits as they are moved to the new branch



```
git checkout feature
git rebase -i main

pick 33d5b7a Message for commit #1
fixup 9480b3d Message for commit #2
pick 5c67e61 Message for commit #3
```

🌳 Rebase – Exploring Branches

Once you understand what rebasing is, the most important thing to learn is **when not to do it.** The golden rule of git rebase is to **never use it on public branches**.



🌳 Cheat Sheets – Exploring Branches

https://education.github.com/git-cheat-sheet-education.pdf

💪 Exercise 2

Task	Expectation
Initializing a Repository (can be javascript/typescript): Create a new directory on your computer, initialize it as a Git repository, and add some files to it.	Understand the initial setup process for a Git repository
Make changes to the files in your repository, stage them, commit them with descriptive messages and push them to GitHub.	Practice the basic workflow of making changes and committing them to the repository.
Create a new branch in your repository, make some changes, switch back and forth between branches while following the **github flow** strategy.	Understand the purpose of branches and how to work with them
Make changes inside **the New branch** then stage them, commit them with descriptive messages and push them to GitHub.	Understand the basic way to push new branch to github
Make a pull request, review the code and merge the changes	Understand the workflow used on the industry

💪 Exercise 2

Task	Expectation
Write a program that takes a temperature in Fahrenheit as input and converts it to Celsius.	Input: A temperature in Fahrenheit. Expected Output: The temperature converted to Celsius.
Write a code to convert centimeter to kilometer or vice versa	Example : 100000 → “1 km”
Write a function that takes an integer n as input and returns true if n is odd and false if n is even	Example : 1000 → isEven: true; 1001→ isEven: false
Write a code to remove the first occurrence of a given “search string” from a string	Example : string = “Hello world”, search string = “ell” → “Ho world”
Write a code to check whether a string is a palindrome or not.	Example : ‘madam’ → palindrome

Feedback

s.id

Thank You !