Git and GitHub

Gain a solid understanding of both Git and GitHub, two essential tools for modern-day developers.

Introduction

Greetings, aspiring developers!

Are you eager to learn about Git and GitHub? Look no further than this guide, which offers detailed explanations of their origins, importance in software development, and key terms and concepts.

But that's not all — you'll also discover practical tips for using Git and GitHub effectively. Learn how to create and manage repositories, collaborate with other developers on code, and more.

Whether you're a beginner or an experienced developer seeking to expand your knowledge, this guide has something for you. So don't delay — start exploring the exciting world of Git and GitHub today with this comprehensive guide!

What is GIT?

GIT is a version control system that allows developers to keep track of changes made to code over time.

It was created by Linus Torvalds in 2005 as a way to manage the development of the Linux kernel. Since then, it has become one of the most popular version control systems in use today.

GIT uses a distributed architecture, which means that every developer working on a project has their own copy of the code repository. This makes it easy for developers to work independently and collaborate on code changes.

What is GitHub?

GitHub is a web-based hosting service for GIT repositories. It was created in 2008 by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett as a way to make GIT repositories more accessible to developers.

GitHub is now the largest host of source code in the world, with millions of repositories and millions of users.

GitHub provides a variety of features to help developers **collaborate** on code, including **pull requests**, **issues**, and **code reviews**. It also has a social aspect, with users able to follow each other, star repositories, and **contribute** to open source projects.

Why is GitHub important?

GitHub has become an essential tool for developers, as it offers many benefits, such as:

- **Collaboration**: GitHub provides an easy way for developers to collaborate on code, work on different branches of a project, and merge their changes together.
- **Version Control**: GitHub uses Git, which allows developers to keep track of changes to code over time, and easily revert to earlier versions if needed.
- **Open Source**: GitHub is home to millions of open source projects, allowing developers to contribute to projects they care about and learn from other developers' code.
- **Portfolio**: GitHub provides a way for developers to showcase their work, build their portfolio, and demonstrate their skills to potential employers.

Key Concepts in GitHub

Here are some key terms you should know when using GitHub:

- **Repository**: A repository is a collection of files and folders that make up a project. Repositories can be either public or private, and can be owned by individuals or organizations.
- **Branch**: A branch is a copy of a repository where changes can be made independently of the main repository. Branches are often used for feature development, bug fixes, or experimental changes.
- **Commit**: A commit is a snapshot of the changes made to a repository at a specific point in time. Each commit has a unique identifier, and includes a message describing the changes made.
- **Pull Request**: A pull request is a proposed change to a repository that is submitted by a developer. Pull requests can be reviewed by other developers, and merged into the main repository if they are accepted.
- **Fork**: A fork is a copy of a repository that is created by a developer. Forks are often used to make changes to a project without affecting the original repository.

Collaboration on GitHub

Here are the key steps to collaborate on GitHub:

Step 1: Fork the repository

If you want to contribute to an existing project on GitHub, the first step is to create your own copy of the project by forking the repository. This creates a new copy of the project in your own GitHub account, which you can modify as needed.

Step 2: Clone the repository

Once you've forked the repository, you need to clone it to your local machine. This creates a local copy of the project that you can work on using your preferred text editor or IDE.

Step 3: Make changes and commit them

Once you've cloned the repository, you can make changes to the code, documentation, or other files as needed. Once you've made changes, you should commit them to your local repository with a descriptive commit message.

Step 4: Push changes to your fork

After committing your changes, you need to push them to your fork on GitHub. This makes your changes available for others to see and review.

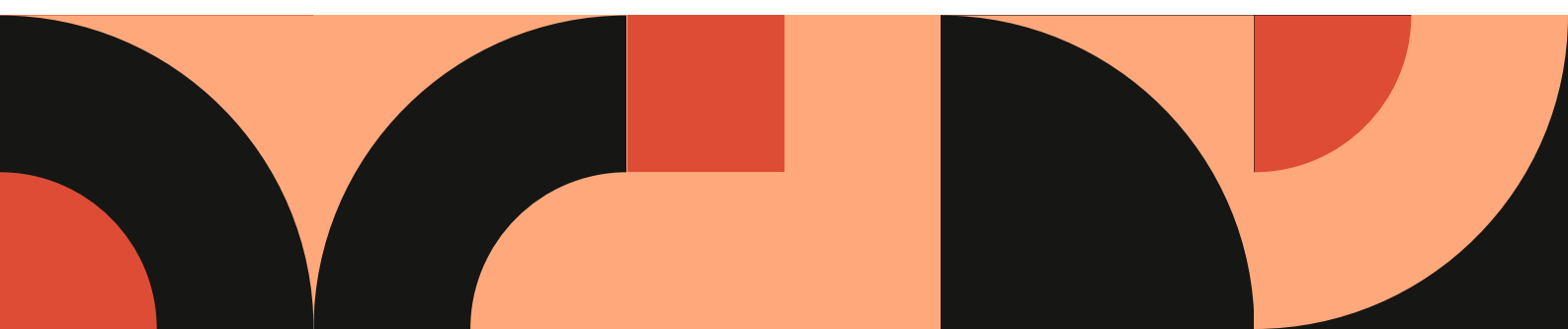
Step 5: Create a pull request

Once you've pushed your changes to your fork, you can create a pull request to merge your changes into the original repository. The pull request will be reviewed by the repository's maintainers, who can then decide whether or not to accept your changes.

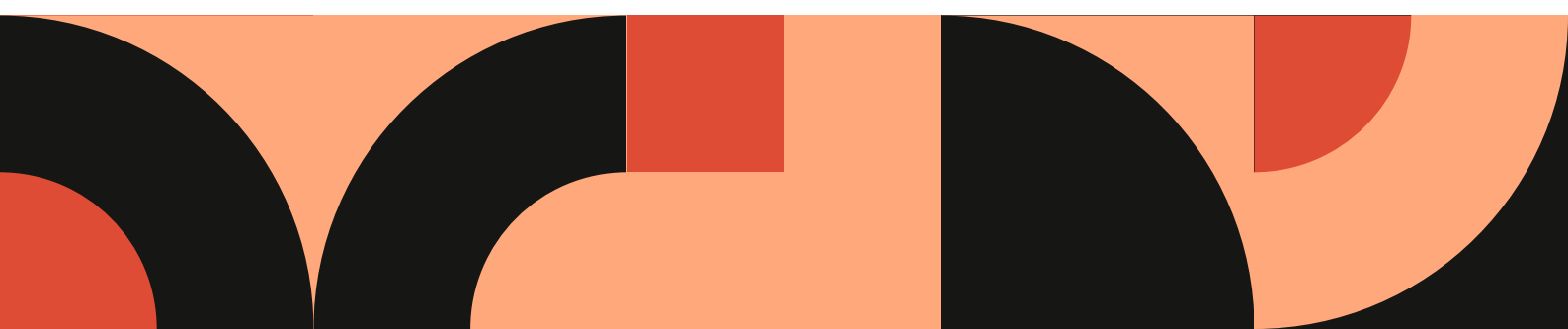
Step 6: Collaborate and resolve conflicts

If your pull request is accepted, you can continue collaborating with other developers on the project. If there are conflicts between your changes and those made by other developers, you'll need to resolve those conflicts before your changes can be merged into the project.

That's it! By following these steps, you can collaborate effectively with other developers on GitHub.



Commonly used Git Commands



Getting Started

- **git init**: Initializes a new Git repository in the current directory.
- **git clone [repository_url]**: Creates a copy of an existing Git repository in a new directory.

Branches

- **git branch**: Lists all local branches in the repository.
- **git branch [branch_name]**: Creates a new branch with the specified name.
- **git checkout [branch_name]**: Switches to the specified branch.
- **git merge [branch_name]**: Merges the specified branch into the current branch.

Pushing and Pulling

- **git remote add [remote_name] [repository_url]**: Adds a new remote repository with the specified name and URL.
- **git push [remote_name] [branch_name]**: Pushes the specified branch to the remote repository.
- **git pull [remote_name] [branch_name]**: Pulls the specified branch from the remote repository and merges it into the local branch.

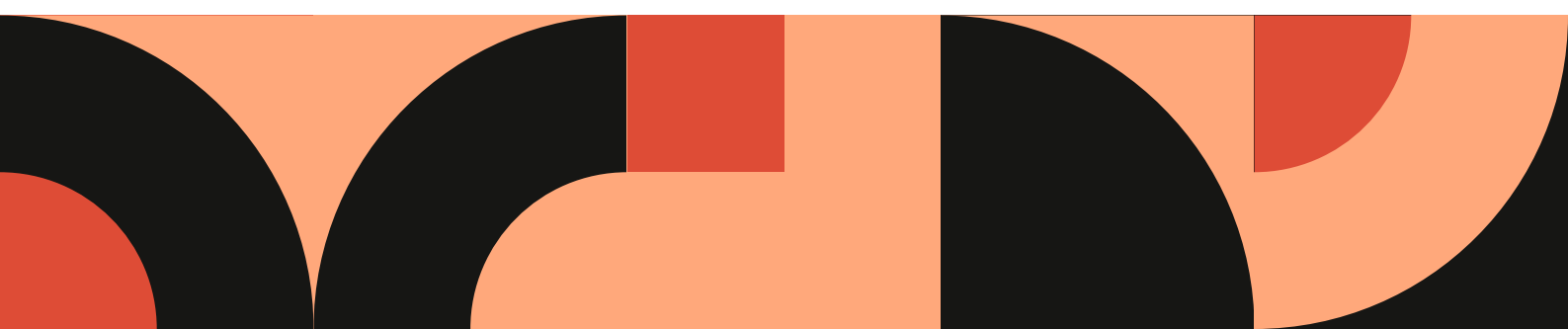
Miscellaneous

- **git status**: Shows the status of the repository, including any changes that have been made.
- **git log**: Shows a log of all commits that have been made to the repository.
- **git diff**: Shows the difference between the current state of the repository and the last commit.

These are just a few of the many commands available in Git. For more information, you can check the Git documentation or run **“git --help”**

Git

Branching

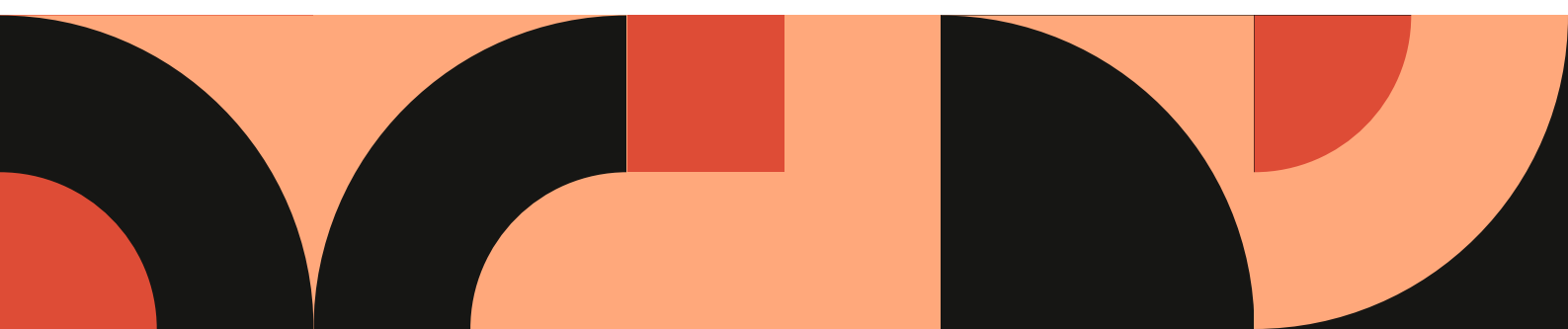


Git branching is the practice of creating separate lines of development within a Git repository. A branch is essentially a separate timeline of changes to the codebase that can be developed independently of the main branch, or "master" branch.

When you create a branch, you create a copy of the codebase at a particular point in time. You can then make changes to that branch without affecting the code in the master branch. This allows developers to work on different features or bug fixes simultaneously without interfering with each other's work.

Once changes are made to a branch, they can be merged back into the main branch when the work is completed and reviewed. This allows changes to be incorporated into the codebase while minimizing the risk of conflicts or errors.

Git branching is a powerful tool for managing code development, especially in larger projects with multiple contributors. It enables teams to work collaboratively and efficiently while maintaining a high level of code quality and organization.



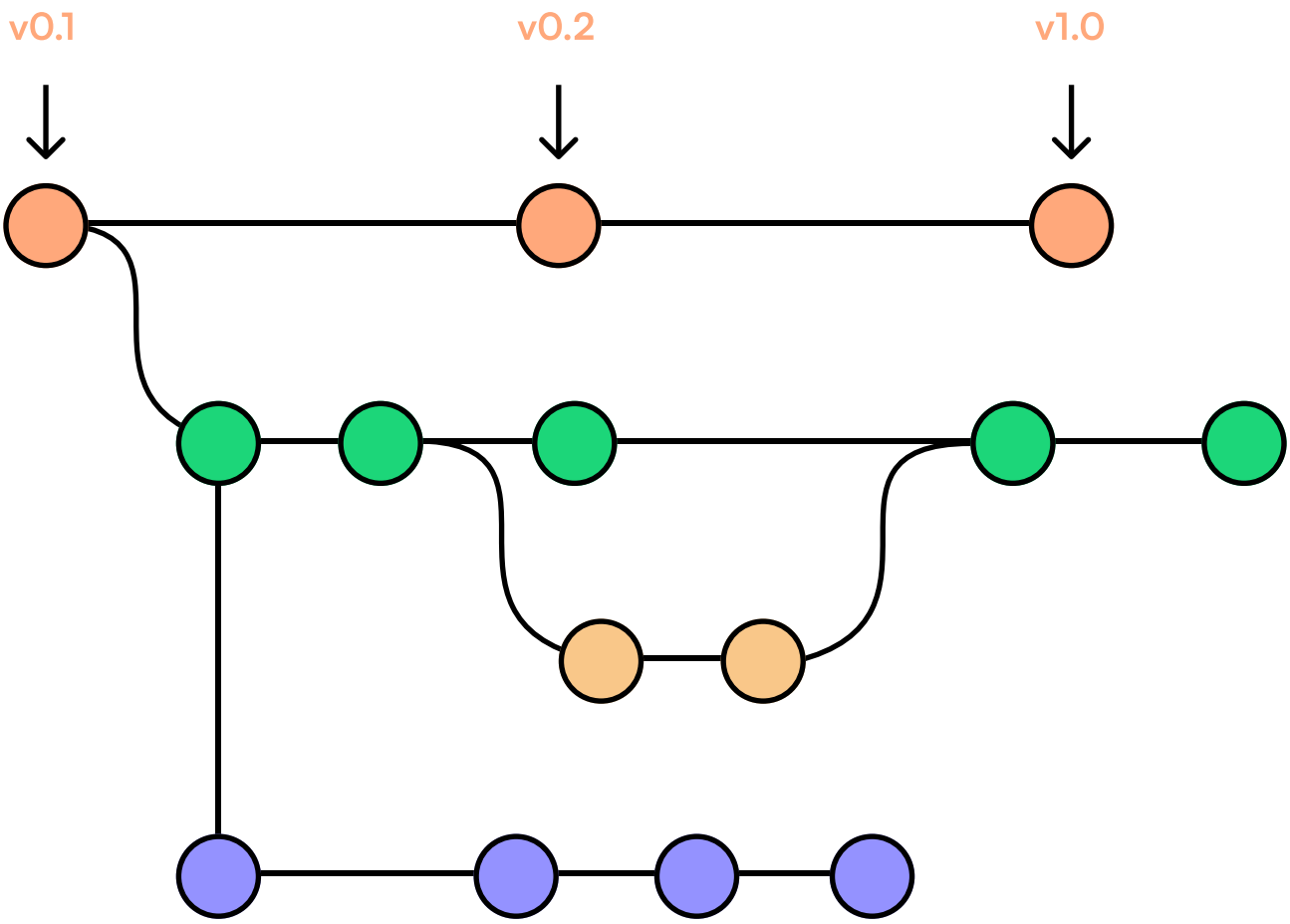
Example

master

dev

testing

feature



Thank You