URL: https://docs.chaicode.com/youtube/chai-aur-git/branches/

Branches in Git | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

On this page

Branches in Git

Branches are a way to work on different versions of a project at the same time. They allow you to create a separate line of development that can be worked on independently of the main branch. This can be useful when you want to make changes to a project without affecting the main branch or when you want to work on a new feature or bug fix.

Some developers can work on Header, some can work on Footer, some can work on Content, and some can work on Layout. This is a good example of how branches can be used in git.

HEAD in git

The HEAD is a pointer to the current branch that you are working on. It points to the latest commit in the current branch. When you create a new branch, it is automatically set as the HEAD of that branch.

the default branch used to be master, but it is now called main. There is nothing special about main, it is just a convention.

Creating a new branch

To create a new branch, you can use the following command:

Terminal window

git

branch

git

branch

bug-fix

git

switch

bug-fix

git

log

git

switch

main

git

switch

-c

dark-mode

git

checkout

orange-mode

Some points to note:

git branch

- This command lists all the branches in the current repository.

git branch bug-fix

- This command creates a new branch called

bug-fix

.

git switch bug-fix

- This command switches to the

bug-fix

branch.

git log

- This command shows the commit history for the current branch.

git switch main

- This command switches to the

main

branch.

git switch -c dark-mode

- This command creates a new branch called

dark-mode

. the

-c

flag is used to create a new branch.

git checkout orange-mode

- This command switches to the

orange-mode

branch.

Commit before switching to a branch

Go to .git folder and checkout to the HEAD file

Merging branches

Merging is about bringing changes from one branch to another.

In Git we have two types of merges :

Fast-Forward Merges (If branches have not diverged)

3-Way Merges (if branches have diverged)

Fast-forward merge

This one is easy as branch that you are trying to merge is usually ahead and there are no conflicts.

When you are done working on a branch, you can merge it back into the main branch. This is done using the following command:

Terminal window

git

checkout

main

git

merge

bug-fix

Some points to note:

git checkout main

- This command switches to the

main

branch.

git merge bug-fix

- This command merges the

bug-fix

branch into the

main

branch.

This is a fast-forward merge. It means that the commits in the

bug-fix

branch are directly merged into the

main

branch. This can be useful when you want to merge a branch that has already been pushed to the remote repository.

3 Way merge

In this type of merge, the main branch has additional commits that are not present in the

bug-fix

branch. This is not a fast-forward merge. Here git looks at 3 different commits [common ancestor of branches + tips of each branch] and combines the changes into one merge commit.

When you are done working on a branch, you can merge it back into the main branch. This is done using the following command:

Terminal window

git

checkout

main

git

merge

bug-fix

If the command are same, what is the difference between fast-forward and not fast-forward merge?

The difference is resolving the conflicts. In a fast-forward merge, there are no conflicts. But in a not fast-forward merge, there are conflicts, and there are no shortcuts to resolve them.

You have to manually resolve the conflicts. Decide, what to keep and what to discard. VSCode has a built-in merge tool that can help you resolve the conflicts.

Managing conflicts

There is no magic button to resolve conflicts. You have to manually resolve the conflicts. Decide, what to keep and what to discard. VSCode has a built-in merge tool that can help you resolve the conflicts. I personally use VSCode merge tool. Github also has a merge tool that can help you resolve the conflicts but most of the time I handle them in VSCode and it gives me all the options to resolve the conflicts.

Overall it sounds scary to beginners but it is not, it?s all about communication and understanding the code

situation with your team members.

Rename a branch

You can rename a branch using the following command:

Terminal window

git

branch

-m

<old-branch-name>

<new-branch-name>

Delete a branch

You can delete a branch using the following command:

Terminal window

git

branch

-d

<branch-name>

Checkout a branch

You can checkout a branch using the following command:

Terminal window

git

checkout

<branch-name>

Checkout a branch means that you are going to work on that branch. You can checkout any branch you want.

List all branches

You can list all branches using the following command:

Terminal window

git

branch

List all branches means that you are going to see all the branches in your repository.

Conclusion

In this section, we have learned about the different types of merges and how to resolve conflicts. We have also learned about the importance of branching and merging in Git and Github.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Behind the scenes

Next

Diff, Stash, Tags

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/branches/#_top

Branches in Git | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

HEAD in git

Creating a new branch

Merging branches

Fast-forward merge

3 Way merge

Managing conflicts

Rename a branch

Delete a branch

Checkout a branch

List all branches

Conclusion

On this page

Overview

HEAD in git

Creating a new branch

Merging branches

Fast-forward merge

3 Way merge

Managing conflicts

Rename a branch

Delete a branch

Checkout a branch

List all branches

Conclusion

Branches in Git

Branches are a way to work on different versions of a project at the same time. They allow you to create a separate line of development that can be worked on independently of the main branch. This can be useful when you want to make changes to a project without affecting the main branch or when you want to work on a new feature or bug fix.

Some developers can work on Header, some can work on Footer, some can work on Content, and some can work on Layout. This is a good example of how branches can be used in git.

HEAD in git

The HEAD is a pointer to the current branch that you are working on. It points to the latest commit in the current branch. When you create a new branch, it is automatically set as the HEAD of that branch.

the default branch used to be master, but it is now called main. There is nothing special about main, it is just a convention.

Creating a new branch

To create a new branch, you can use the following command:

Terminal window

git

branch

git

branch

bug-fix

git

switch

bug-fix

git

log

git

switch

main

git

switch

-c

dark-mode

git

checkout

orange-mode

Some points to note:

git branch

- This command lists all the branches in the current repository.

git branch bug-fix

- This command creates a new branch called

bug-fix

.

git switch bug-fix

- This command switches to the

bug-fix

branch.

git log

- This command shows the commit history for the current branch.

git switch main

- This command switches to the

main

branch.

git switch -c dark-mode

- This command creates a new branch called

dark-mode

. the

-c

flag is used to create a new branch.

git checkout orange-mode

- This command switches to the

orange-mode

branch.

Commit before switching to a branch

Go to .git folder and checkout to the HEAD file

Merging branches

Merging is about bringing changes from one branch to another.

In Git we have two types of merges :

Fast-Forward Merges (If branches have not diverged)

3-Way Merges (if branches have diverged)

Fast-forward merge

This one is easy as branch that you are trying to merge is usually ahead and there are no conflicts.

When you are done working on a branch, you can merge it back into the main branch. This is done using the

following command:

Terminal window

git

checkout

main

git

merge

bug-fix

Some points to note:

git checkout main

- This command switches to the

main

branch.

git merge bug-fix

- This command merges the

bug-fix

branch into the

main

branch.

This is a fast-forward merge. It means that the commits in the

bug-fix

branch are directly merged into the

main

branch. This can be useful when you want to merge a branch that has already been pushed to the remote repository.

3 Way merge

In this type of merge, the main branch has additional commits that are not present in the

bug-fix

branch. This is not a fast-forward merge. Here git looks at 3 different commits [common ancestor of branches

+ tips of each branch] and combines the changes into one merge commit.

When you are done working on a branch, you can merge it back into the main branch. This is done using the following command:

Terminal window

git

checkout

main

git

merge

bug-fix

If the command are same, what is the difference between fast-forward and not fast-forward merge?

The difference is resolving the conflicts. In a fast-forward merge, there are no conflicts. But in a not fast-forward merge, there are conflicts, and there are no shortcuts to resolve them.

You have to manually resolve the conflicts. Decide, what to keep and what to discard. VSCode has a built-in merge tool that can help you resolve the conflicts.

Managing conflicts

There is no magic button to resolve conflicts. You have to manually resolve the conflicts. Decide, what to keep and what to discard. VSCode has a built-in merge tool that can help you resolve the conflicts. I personally use VSCode merge tool. Github also has a merge tool that can help you resolve the conflicts but most of the time I handle them in VSCode and it gives me all the options to resolve the conflicts.

Overall it sounds scary to beginners but it is not, it?s all about communication and understanding the code situation with your team members.

Rename a branch

You can rename a branch using the following command:

Terminal window

git

branch

-m

<old-branch-name>

<new-branch-name>

Delete a branch

You can delete a branch using the following command:

Terminal window

git

branch

-d

<branch-name>

Checkout a branch

You can checkout a branch using the following command:

Terminal window

git

checkout

<branch-name>

Checkout a branch means that you are going to work on that branch. You can checkout any branch you want.

List all branches

You can list all branches using the following command:

Terminal window

git

branch

List all branches means that you are going to see all the branches in your repository.

Conclusion

In this section, we have learned about the different types of merges and how to resolve conflicts. We have also learned about the importance of branching and merging in Git and Github.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Behind the scenes

Next

Diff, Stash, Tags

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/

Home | Chai aur Docs

Skip to content

Chai aur Docs

YouTube

Instagram

LinkedIn

GitHub

X

Docs You?ll

Actually Read

Next-gen docs that builds reading habits into your workflow.

Start Learning

Watch Tutorials

Have a question or want to get involved..?

Join our Discord

Brought to you by

ChaiCode

Chai aur Code is an unique initiative by Hitesh Choudhary where he mentors people who want to learn programming and grow in the field.

Learn about ChaiCode

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/getting-started/

Getting Started | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Maximize Your Learning

On this page

Overview

Maximize Your Learning

Getting Started

Reading the docs is a great way to learn. Whether it?s a new technology, programming language, or framework, delving into the docs helps you gain in-depth knowledge and insights.

We have designed this website so that you can develop the habit to read from the docs. In future, our attempt is to provide you guide with all videos so that you can learn directly from docs and get all information in one place.

No need to make notes or write down anything. Just read the docs.

Maximize Your Learning

Read Actively:

Take Your Time: Don?t rush through the content. Take the time to understand each section thoroughly.

Highlight Key Points: If you find something important, highlight it or make a note of it for future reference.

Practice What You Learn

Hands-On Practice: Try out code examples and exercises as you read through the documentation. This will help reinforce your understanding.

Build Projects: Apply what you?ve learned by building small projects or components. This practical application is crucial for mastering new skills.

Utilize Additional Resources

Cross-Reference: If a topic is unclear, look for additional resources like blogs, videos, or forums for different explanations and perspectives.

Ask Questions: Don?t hesitate to ask questions in our community or seek help from peers if you encounter any difficulties.

Stay Organized

Bookmark Important Sections: Use bookmarks to keep track of important sections or topics you may want to revisit.

Use the Search Feature: Make use of the search functionality to quickly find specific information.

Engage with the Community

Join Discussions: Participate in community discussions and forums to share knowledge and gain insights from others.

Contribute: If you find any errors or have suggestions for improvements, consider contributing to the docs to help others.

By following these tips, you can maximize your learning experience and make the most out of the documentation provided. Happy learning!.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Chaudhary

Last updated:

Apr 22, 2025

Next

Welcome

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-html/welcome/

Welcome | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

On this page

Overview

Welcome

Haanji! Swagat hai

Chai aur Docs

mein. ?

This guide has been carefully curated as a comprehensive reference for the ?

Chai aur HTML

? series on the

Chai aur Code

YouTube channel. For the best learning experience, we recommend following these docs alongside our video tutorials.

Play

Let?s begin with the basics in the next section?

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Getting Started

Next

HTML Intro

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-html/introduction/

Introduction to HTML | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

What is HTML?

What is HTML5?

How much HTML should you learn?

Text/Code Editor vs. Word Processor

Recommended Code Editors

Essential VS Code Extensions for HTML

Emmet for HTML Productivity

On this page

Overview

What is HTML?

What is HTML5?

How much HTML should you learn?

Text/Code Editor vs. Word Processor

Recommended Code Editors

Essential VS Code Extensions for HTML

Emmet for HTML Productivity

Introduction to HTML

HTML is the foundation of all websites.

In this guide, you?ll understand the essentials of HTML and get started with web development.

What is HTML?

HTML stands for

HyperText Markup Language

. It?s the standard markup language for creating and structuring web pages. HTML defines the structure and content of your web page, such as headings, paragraphs, images, and links.

What is HTML5?

HTML5

is the latest version of HTML. It introduces new tags, attributes, and features, making it simpler and more efficient to build modern, interactive web pages. HTML5 is also designed to be fully backward compatible with previous HTML versions.

How much HTML should you learn?

You only need the basics of HTML to start creating websites. Typically, mastering HTML essentials should not take more than a weekend. Understand the core tags and structure, and you?re all set to build web pages effectively.

Text/Code Editor vs. Word Processor

Text editors

are specialized tools for writing and editing plain text files, including source code. They include essential features for coding such as syntax highlighting, auto-completion, and code formatting.

Word processors

like Microsoft Word or Google Docs are ideal for creating formatted documents such as letters, reports, or presentations?but they?re not suitable for coding.

For HTML development, always use a

code editor

.

Recommended Code Editors

Here are some recommended code editors for writing HTML efficiently:

Visual Studio Code

? Powerful, widely-used, and beginner-friendly.

Zed

? Fast and collaborative editor for modern development.

VIM

? Highly customizable and efficient (but notoriously tricky to exit!).

Helix

? Similar to VIM, but easier to use right out of the box.

Essential VS Code Extensions for HTML

Enhance your HTML coding experience with these recommended VS Code extensions:

HTML Snippets

? Quickly insert common HTML structures.

Live Server

? Automatically refreshes your browser as you edit your HTML.

Emmet for HTML Productivity

Emmet is built-in with VS Code and allows you to rapidly generate HTML code using short abbreviations.

This significantly speeds up your coding workflow. No need to manually type lengthy tags?let Emmet handle it for you.

Spend some time getting familiar with Emmet shortcuts to greatly improve your productivity. Learn more about Emmet at the

official website

.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Welcome

Next

Emmet Crash Course

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-html/emmit-crash-course/

Emmet Crash Course | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Some common Emmit shortcuts

ID and Class

Grouping

CSS shortcuts

Conclusion

On this page

Overview

Some common Emmit shortcuts

ID and Class

Grouping

CSS shortcuts

Conclusion

Emmet Crash Course

Emmit is a code snippets manager for VS Code. It is used to create HTML code faster. Emmit is a must-have tool for any web developer. In VS Code, Emmit is enabled by default. It works only after you have created a new HTML file.

Learn the shortcuts and just press the tab or enter key to get the code you want.

Some common Emmit shortcuts

!

- Inserts a

<!DOCTYPE html>

tag

h1

- Inserts a

<h1>

tag

h2

- Inserts a

<h2>

tag

p

- Inserts a

<p>

tag

img

- Inserts an

tag

a

- Inserts an

<a>

tag

ul

- Inserts an

tag

ul>li

- Inserts a

tag inside an

tag

ul>li>a

- Inserts an

<a>

tag inside a

tag inside an

tag

ul>li*3

- Inserts 3

tags inside an

tag

div

- Inserts a

tag

div>p

- Inserts a

<p>

tag inside a

tag

div>p*3

- Inserts 3

<p>

tags inside a

tag

ID and Class

#

- Inserts an

id

attribute

.

- Inserts a

class

attribute

Example:

#my-id

- Inserts an

id

attribute with the value

my-id

.my-class

- Inserts a

class

attribute with the value

my-class

div>(header>ul>li*2>a)+footer>p1

- expands into

<

div

>

<

header

>

<

ul

>

<

li

><

a

href

=

""

></

a

></

li

>

<

li

><

a

href

=

""

></

a

></

li

>

</

ul

>

</

header

>

<

footer

>

<

p

></

p

>

</

footer

>

</

div

>

Grouping

div>(header>ul>li*2>a)+footer>p

- Inserts a

tag with a

<header>

tag inside it, a

tag inside it, and 2

tags inside the

tag. Then it inserts an

<a>

tag inside each

tag. Finally it inserts a

<footer>

tag and a

<p>

tag inside it.

Yep, it can go little bit crazy. But you don?t have to worry about it. Rarely you will need to use it.

CSS shortcuts

style

- Inserts a

<style>

tag

pos

- Inserts a

position

property

pos:absolute

- Inserts a

position

property with the value

absolute

bgc

- Inserts a

background-color

property

bgc:red

- Inserts a

background-color

property with the value

red

ma

- Inserts a

margin:auto

property

Conclusion

Emmit is a must-have tool for any web developer. BUT this does not mean you have to learn every single shortcut. You can use Emmit to create HTML code faster. NO ONE remembers all the shortcuts. We use then mostly by trial and error, and learning them as we go.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

HTML Intro

Next

Common HTML Tags

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-html/html-tags/

Common HTML Tags | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Basic Terminology

HTML Tags for Text Content

HTML Tags for Lists

HTML Tags for Tables

HTML Tags for Forms

HTML Tags for Media

HTML Tags for Linking and Metadata

Script Tag Variations

HTML Tags for Semantic and Meta Content

Attributes for HTML Tags

HTML5 attributes

HTML5 tags

Conclusion

On this page

Overview

Basic Terminology

HTML Tags for Text Content

HTML Tags for Lists

HTML Tags for Tables

HTML Tags for Forms

HTML Tags for Media

HTML Tags for Linking and Metadata

Script Tag Variations

HTML Tags for Semantic and Meta Content

Attributes for HTML Tags

HTML5 attributes

HTML5 tags

Conclusion

Common HTML Tags

Focus on the Essentials

Remember, you don?t need to master HTML to become a web developer. Focus on the basics and move on

quickly. HTML5 only adds a few new tags and attributes to the classic HTML vocabulary. Although accessibility is very important, we?ll cover that later?especially if you plan to build web applications with JavaScript or modern frameworks.

Basic Terminology

Tag

? A piece of text enclosed in angle brackets (e.g.,

<p>

) that defines an HTML element.

Attribute

? Extra information provided inside an opening tag to modify an element (e.g.,

id="main"

).

Element

? A complete HTML structure, typically including an opening tag, content, and a closing tag.

HTML Tags for Text Content

<p>

? Paragraph

? Inline container for text

? Generic container (block-level element)

<a>

? Anchor (hyperlink)

? Image

? Line break

<hr>

? Horizontal rule

<b>

? Bold text

<i>

? Italic text

<u>

? Underlined text

? Strong importance (semantically bold)

? Emphasized text (typically italicized)

<code>

? Code snippet

<pre>

? Preformatted text

HTML Tags for Lists

? Unordered list

? Ordered list

? List item

HTML Tags for Tables

<table>

? Table container

<tr>

? Table row

<td>

? Table cell

HTML Tags for Forms

<form>

? Form container

<input>

? Input field

<textarea>

? Multi-line text input

<select>

? Drop-down list

<option>

? Option within a drop-down

<button>

? Button

HTML Tags for Media

? Image element

<source>

? Media resource (used within

<picture>

or

<video>

)

<picture>

? Container for multiple image sources

<video>

? Video element

<audio>

? Audio element

HTML Tags for Linking and Metadata

<link>

? Links external resources (e.g., CSS files, favicons)

<meta>

? Provides metadata about the document

<script>

? Embeds or references executable scripts

Script Tag Variations

<script src="script.js"></script>

? External script

<script async src="script.js"></script>

? Asynchronously loaded script

<script defer src="script.js"></script>

? Deferred script execution

<script type="module" src="script.js"></script>

? JavaScript module

HTML Tags for Semantic and Meta Content

Modern HTML (HTML5) introduces several semantic tags that enhance the meaning and structure of your document:

<header>

? Defines a header for a document or section

<footer>

? Defines a footer for a document or section

<nav>

? Defines navigation links

<main>

? Specifies the main content of a document

<article>

? Encloses self-contained content

<section>

? Groups related content together

<aside>

? Represents content aside from the main content

<details>

? Defines additional details the user can view or hide

<summary>

? Provides a summary for the

<details>

element

<time>

? Represents a specific period in time

Additionally, standard meta tags include:

<meta charset="utf-8">

? Sets the character encoding

<meta name="viewport" content="width=device-width, initial-scale=1.0">

? Ensures responsiveness

<meta name="description" content="Description">

? Provides a page description

<meta name="author" content="Author">

? Identifies the author

<meta name="keywords" content="Keywords">

? Supplies SEO keywords

<meta name="robots" content="index, follow">

? Guides search engine indexing

<meta name="googlebot" content="index, follow">

? Specific for Googlebot

Attributes for HTML Tags

Attributes provide additional information for HTML elements. Here are some common examples:

<

p

id

=

"

my-id

"

class

=

"

my-class

"

>

Hello World

</

p

>

<

img

src

=

"

image.jpg

"

alt

=

"

Descriptive image text

"

>

<

a

href

=

"

https://www.google.com

"

>

Visit Google

</

a

>

<

input

type

=

"

text

"

placeholder

=

"

Enter your name

"

>

<

button

>

Click me

</

button

>

id="my-id"

- Adds an

id

attribute with the value

my-id

.my-class

- Adds a

class

attribute with the value

my-class

src="image.jpg"

- Adds a

src

attribute with the value

image.jpg

alt="Image"

- Adds an

alt

attribute with the value

Image

href="https://www.google.com"

- Adds a

href

attribute with the value

https://www.google.com

type="text"

- Adds a

type

attribute with the value

text

placeholder="Enter your name"

- Adds a

placeholder

attribute with the value

Enter your name

Some attributes are global attributes and can be used on any HTML tag. Some attributes are specific to

certain tags and can only be used with that tag. For example, the

href

attribute is a specific attribute for the

<a>

tag, and the

title

attribute is a global attribute that can be used on any HTML tag.

HTML5 attributes

Some examples of HTML5 attributes are:

autofocus

- Adds an

autofocus

attribute to an input field

required

- Adds a

required

attribute to an input field

readonly

- Adds a

readonly

attribute to an input field

section

- Adds a

section

attribute to a section element

footer

- Adds a

footer

attribute to a footer element

footer, section, and header are new HTML5 attributes. They are used to define the structure of a web page.

Fundamentally, they are used to group related content together, just like the

tag is used to group related content together.

HTML5 tags

<header>

- Header

<footer>

- Footer

<nav>

- Navigation

<main>

- Main

<article>

- Article

<section>

- Section

<aside>

- Aside

<details>

- Details

<summary>

- Summary

<time>

- Time

<mark>

- Mark

<meter>

- Meter

<progress>

- Progress

<video>

- Video

<audio>

- Audio

<source>

- Source

Conclusion

You don?t need to do a PhD in HTML to be a web developer. You just need to know the basics and get out of here ASAP. HTML5 just adds a few new tags and attributes to HTML. Rest all the stuff like Web APIs (local storage, session storage, etc.) are just JavaScript stuff. You can learn them later.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Emmet Crash Course

Next

Welcome

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/welcome/

Welcome | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

On this page

Overview

Welcome

Haanji! Swagat hai

Chai aur Docs

mein. ?

This guide has been carefully curated as a comprehensive reference for the ?

Chai aur Git

? series on the

Chai aur Code

YouTube channel. For the best learning experience, we recommend following these docs alongside our video tutorials.

Play

Let?s begin with the basics in the next section?

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Common HTML Tags

Next

Git and GitHub

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/introduction/

Git and GitHub | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Git and Github are different

A little on version control systems

Learning Path

Install Git

Account on Github

Conclusion

On this page

Overview

Git and Github are different

A little on version control systems

Learning Path

Install Git

Account on Github

Conclusion

Git and GitHub

Let?s start with the basics. Git is a version control system that allows you to track changes to your files and

collaborate with others. It is used to manage the history of your code and to merge changes from different branches. I can understand that as of now these terms like version control, branches, and merges are not familiar to you. But don?t worry, we will learn them in this tutorial.

Git and Github are different

Git is a version control system that is used to track changes to your files. It is a free and open-source software that is available for Windows, macOS, and Linux. Remember, GIT is a software and can be installed on your computer.

Github is a web-based hosting service for Git repositories. Github is an online platform that allows you to store and share your code with others. It is a popular platform for developers to collaborate on projects and to share code. It is not that Github is the only provider of Git repositories, but it is one of the most popular ones.

A little on version control systems

Version control systems are used to manage the history of your code. They allow you to track changes to your files and to collaborate with others. Version control systems are essential for software development. Consider version control as a checkpoint in game. You can move to any time in the game and you can always go back to the previous checkpoint. This is the same concept in software development.

Before Git became mainstream, version control systems were used by developers to manage their code. They were called SCCS (Source Code Control System). SCCS was a proprietary software that was used to manage the history of code. It was expensive and not very user-friendly. Git was created to replace SCCS and to make version control more accessible and user-friendly. Some commong version control systems are Subversion (SVN), CVS, and Perforce.

Learning Path

In this tutorial, we will learn the basics of Git and Github. We will start with the basics and then move on to more advanced topics. We will also learn how to use Git and Github for collaboration and version control. By the end of this tutorial, you will have a good understanding of Git and Github and will be able to use them to manage your code effectively.

We will go in this jounney something like this:

Get the basics

Use it daily

Face the problems

Solve them

Learn more

We will focus more on Git first, once you understand git, moving towards Github will be easy.

Install Git

To install Git, you can use command line or you can visit official website and download the installer for your operating system. Git is available for Windows, macOS, and Linux and is available at

https://git-scm.com/downloads

.

Account on Github

Another step that you have to follow is to create an account on Github. I will later walk you through the process of linking your Github account with your Machine. You cannot push your code to Github without ssh-key setup. Password authentication is not recommended and these days it is not possible to use it. So, you need to setup ssh-key authentication. We will cover that in a later part of the tutorial.

Conclusion

In this part, we have learned the basics of Git and Github. We have also learned how to install Git and Github. We have also learned about the importance of version control systems and how they are used in software development.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Welcome

Next

Terminology

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/terminology/

Terminology | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Check your git version

Repository

Your config settings

Creating a repository

Commit

Complete git flow

Stage

Commit

Logs

change default code editor

gitignore

Conclusion

On this page

Overview

Check your git version

Repository

Your config settings

Creating a repository

Commit

Complete git flow

Stage

Commit

Logs

change default code editor

gitignore

Conclusion

Terminology

Git and people who use it talk in a different terminology. For example they don?t call it a folder, they call it a repository. They don?t call it alternative timeline, they call it branch. Although, I agree that alternative timeline is a better name for it. ?

Check your git version

To check your git version, you can run the following command:

Terminal window

git

--version

This command will display the version of git installed on your system. Git is a very stable software and don?t get any breaking changes in majority of the cases, at least in my experience.

Repository

A repository is a collection of files and directories that are stored together. It is a way to store and manage your code. A repository is like a folder on your computer, but it is more than just a folder. It can contain other files, folders, and even other repositories. You can think of a repository as a container that holds all your code.

There is a difference between a software on your system vs tracking a particular folder on your system. At any point you can run the following command to see the current state of your repository:

Terminal window

git

status

Not all folders are meant to be tracked by git. Here we can see that all green folders are projects are getting tracked by git but red ones are not.

Your config settings

Github has a lot of settings that you can change. You can change your username, email, and other settings. Whenever you checkpoint your changes, git will add some information about your such as your username and email to the commit. There is a git config file that stores all the settings that you have changed. You can make settings like what editor you would like to use etc. There are some global settings and some repository specific settings.

Let?s setup your email and username in this config file. I would recommend you to create an account on github and then use the email and username that you have created.

Terminal window

git

config

--global

user.email

"

"

git

config

--global

user.name

"

Your Name

"

Now you can check your config settings:

Terminal window

git

config

--list

This will show you all the settings that you have changed.

Creating a repository

Creating a repository is a process of creating a new folder on your system and initializing it as a git repository.

It?s just regular folder to code your project, you are just asking git to track it. To create a repository, you can

use the following command:

Terminal window

git

status

git

init

git status

command will show you the current state of your repository.

git init

command will create a new folder on your system and initialize it as a git repository. This adds a hidden

.git

folder to your project.

Commit

commit is a way to save your changes to your repository. It is a way to record your changes and make them permanent. You can think of a commit as a snapshot of your code at a particular point in time. When you commit your changes, you are telling git to save them in a permanent way. This way, you can always go back to that point in time and see what you changed.

Usual flow looks like this:

Complete git flow

A complete git flow, along with pushing the code to github looks like this:

When you want to track a new folder, you first use

init

command to create a new repository. Then you can use

add

command to add the folder to the repository. After that you can use

commit

command to save the changes. Finally you can use

push

command to push the changes to github. Of course there is more to it but this is the basic flow.

Stage

Stage is a way to tell git to track a particular file or folder. You can use the following command to stage a file:

Terminal window

git

init

git

add

<file>

<file2>

git

status

Here we are initializing the repository and adding a file to the repository. Then we can see that the file is now being tracked by git. Currently our files are in staging area, this means that we have not yet committed the changes but are ready to be committed.

Commit

Terminal window

git

commit

-m

"

commit message

"

git

status

Here we are committing the changes to the repository. We can see that the changes are now committed to the repository. The

-m

flag is used to add a message to the commit. This message is a short description of the changes that were

made. You can use this message to remember what the changes were.

Missing the

-m

flag will result in an action that opens your default settings editor, which is usually VIM. We will change this to vscode in the next section.

Logs

Terminal window

git

log

This command will show you the history of your repository. It will show you all the commits that were made to the repository. You can use the

--oneline

flag to show only the commit message. This will make the output more compact and easier to read.

?? - Check git log docs

Atomic commits are a way to make sure that each commit is a self-contained unit of work. This means that if one commit fails, you can always go back to a previous commit and fix the issue. This is important for maintaining a clean and organized history in your repository.

change default code editor

You can change the default code editor in your system to vscode. To do this, you can use the following command:

Terminal window

git

config

--global

core.editor

"

code --wait

"

gitignore

Gitignore is a file that tells git which files and folders to ignore. It is a way to prevent git from tracking certain files or folders. You can create a gitignore file and add list of files and folders to ignore by using the following command:

Example:

.gitignore

node_modules

.env

.vscode

Now, when you run the

git status

command, it will not show the

node_modules

and

.vscode

folders as being tracked by git.

Conclusion

In this section, we have learned about the basics of git and how to use it to track changes to your files and folders. We have also learned about the different commands that you can use to interact with your repository, such as

init

,

add

,

commit

,

log

, etc By the end of this section, you should have a good understanding of how to use git and how to use it effectively to manage your code.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Git and GitHub

Next

Behind the scenes

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/behind-the-scenes/

Git behind the scenes | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Git Snapshots

3 Musketeers of Git

Commit Object

Tree Object

Blob Object

Helpful commands

On this page

Overview

Git Snapshots

3 Musketeers of Git

Commit Object

Tree Object

Blob Object

Helpful commands

Git behind the scenes

Git is a version control system that allows you to track changes to your files and folders. It is a powerful tool that can help you manage your code more effectively. In this section, we will explore the basics of how git works internally.

Git Snapshots

A git snapshot is a point in time in the history of your code. It represents a specific version of your code, including all the files and folders that were present at that time. Each snapshot is identified by a unique hash code, which is a string of characters that represents the contents of the snapshot.

A snapshot is not an image, it?s just a representation of the code at a specific point in time. Snapshot is a loose term that is used when git stores information about the code in a locally stored key-value based database. Everything is stored as an object and each object is identified by a unique hash code.

3 Musketeers of Git

The three musketeers of git are:

Commit Object

Tree Object

Blob Object

Commit Object

Each commit in the project is stored in

.git

folder in the form of a commit object. A commit object contains the following information:

Tree Object

Parent Commit Object

Author

Committer

Commit Message

Tree Object

Tree Object is a container for all the files and folders in the project. It contains the following information:

File Mode

File Name

File Hash

Parent Tree Object

Everything is stored as key-value pairs in the tree object. The key is the file name and the value is the file

hash.

Blob Object

Blob Object is present in the tree object and contains the actual file content. This is the place where the file content is stored.

Helpful commands

Here are some helpful commands that you can use to explore the git internals:

Terminal window

git

show

-s

--pretty=raw

<commit-hash>

Grab tree id from the above command and use it in the following command to get the tree object:

Terminal window

git

ls-tree

<tree-id>

Grab tree id from the above command and use it in the following command to get the blob object:

Terminal window

git

show

<blob-id>

Grab tree id from the above command and use it in the following command to get the commit object:

Terminal window

git

cat-file

-p

<commit-id>

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Terminology

Next

Branches in Git

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/diff-stash-tags/

Diff, Stash and Tags | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Diff, Stash and Tags

This guide will help you understand the different commands related to diff, tags and stash in git. These are not main stream commands but they are very useful in certain situations.

Git diff

The

git diff

is an informative command that shows the differences between two commits. It is used to compare the changes made in one commit with the changes made in another commit. Git consider the changed versions of same file as two different files. Then it gives names to these two files and shows the differences between them.

How to Read the Diff Output

a/

? the original file (before changes)

b/

? the updated file (after changes)

? marks the beginning of the original file

+++

? marks the beginning of the updated file

@@

? shows the line numbers and position of changes

Here the file A and file B are the same file but different versions.

Git will show you the changes made in the file A and file B. It will also show you the line number where the change occurred along with little preview of the change.

Comparing Working Directory and Staging Area

Terminal window

git

diff

This command shows the unstaged changes in your working directory compared to the staging area. This command alone will not show you the changes made in the file A and file B, you need to provide options to show the changes.

Comparing Staging Area with Repository

Terminal window

git

diff

--staged

This command shows the changes between your last commit and the staging area (i.e., changes that are staged and ready to be committed).

Comparing Two Branches

Terminal window

git

diff

<branch-name-one>

<branch-name-two>

This command compares the difference between two branches.

Another way to compare the difference between two branches is to use the following command:

Terminal window

git

diff

branch-name-one..branch-name-two

Comparing Specific Commits:

Terminal window

git

diff

<commit-hash-one>

<commit-hash-two>

This command compares the difference between two commits.

Git Stash

Stash is a way to save your changes in a temporary location. It?s useful when switching branches without losing work. You can then come back to the file later and apply the changes.

Conflicting changes will not allow you to switch branches without committing the changes. Another alternative is to use the

git stash

command to save your changes in a temporary location.

Terminal window

git

stash

This command saves your changes in a temporary location. It is like a stack of changes that you can access later.

Naming the stash

You can also name the stash by using the following command:

Terminal window

git

stash

save

"

work in progress on X feature

"

View the stash list

You can view the list of stashes by using the following command:

Terminal window

git

stash

list

Apply the Most Recent Stash

You can apply the stash by using the following command:

Terminal window

git

stash

apply

Apply Specific Stash

You can apply the specific stash by using the following command:

Terminal window

git

stash

apply

stash@{

0

}

Here

stash@{0}

is the name of the stash. You can use the

git stash list

command to get the name of the stash.

Applying and Drop a Stash

You can apply and drop the stash by using the following command:

Terminal window

git

stash

pop

This command applies the stash and drops it from the stash list.

Drop the stash

You can drop the stash by using the following command:

Terminal window

git

stash

drop

Applying stash to a specific branch

You can apply the stash to a specific branch by using the following command:

Terminal window

git

stash

apply

stash@{

0

}

<branch-name>

Clearing the stash

You can clear the stash by using the following command:

Terminal window

git

stash

clear

Git Tags

Tags are a way to mark a specific point in your repository. They are useful when you want to remember a

specific version of your code or when you want to refer to a specific commit. Tags are like sticky notes that you can attach to your commits.

Creating a tag

You can create a tag using the following command:

Terminal window

git

tag

<tag-name>

This command creates a new tag with the specified name. The tag will be attached to the current commit.

Create an annotated tag

You can create an annotated tag using the following command:

Terminal window

git

tag

-a

<tag-name>

-m

"

Release 1.0

"

This command creates an annotated tag with the specified name and message. The tag will be attached to the current commit.

List all tags

You can list all tags using the following command:

Terminal window

git

tag

This command lists all the tags in your repository.

Tagging a specific commit

You can tag a specific commit using the following command:

Terminal window

git

tag

<tag-name>

<commit-hash>

Push tags to remote repository

You can push tags to a remote repository using the following command:

Terminal window

git

push

origin

<tag-name>

Delete a tag

You can delete a tag using the following command:

Terminal window

git

tag

-d

<tag-name>

Delete tag on remote repository

You can delete a tag on a remote repository using the following command:

Terminal window

git

push

origin

:<tag-name>

Conclusion

In this section, we explored how to use Git?s diff, stash, and tags commands. Though not used as frequently as add, commit, or push, they are incredibly helpful in debugging, context switching, and release management. See you next tutorial.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Branches in Git

Next

Managing History

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/managing-history/

Managing History | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

PostgreSQL on VPS

Advance Node Logger

YouTube

Instagram

LinkedIn

GitHub

X

On this page

Overview

Merge commits

Rebase in git

Ensure you are on the branch you want to rebase

Resolve any conflicts

Git reflog

View the reflog:

Find specific commit

Recover lost commits or changes

Conclusion

On this page

Overview

Merge commits

Rebase in git

Ensure you are on the branch you want to rebase

Resolve any conflicts

Git reflog

View the reflog:

Find specific commit

Recover lost commits or changes

Conclusion

Managing History

This guide will help you understand how to manage your Git history effectively.

Merge commits

A merge commit is a commit that combines two or more commits into one. It is created when you merge two or more branches into a single branch. The merge commit contains all the changes from the original branches, and it is used to keep the project history clean and easy to understand.

Rebase in git

Git rebase is a powerful Git feature used to change the base of a branch. It effectively allows you to move a branch to a new starting point, usually a different commit, by ?replaying? the commits from the original base onto the new base. This can be useful for keeping a cleaner, linear project history.

Some people like to use rebase over the merge command because it allows you to keep the commit history cleaner and easier to understand. It also allows you to make changes to the code without affecting the original branch.

Here?s a flow example of using git rebase with all the commands involved:

Suppose you have a feature branch called feature-branch that you want to rebase onto the main branch.

Ensure you are on the branch you want to rebase

Terminal window

git

checkout

feature-branch

git

rebase

main

This will replay the commits from feature-branch on top of the latest changes in main.

Resolve any conflicts

If there are any conflicts, you will need to resolve them manually. You can use the merge tool in VSCode to resolve the conflicts.

Terminal window

git

add

<resolved-files>

git

rebase

--continue

Try to avoid ?force option when using rebase. It can cause issues with the project history. I have seen many horror stories of people using ?force to fix conflicts.

Git reflog

Git reflog is a command that shows you the history of your commits. It allows you to see the changes that you have made to your repository over time. This can be useful for debugging and understanding the history of your project.

View the reflog:

Terminal window

git

reflog

This will show you the history of your commits. You can use the number at the end of each line to access the commit that you want to view.

Find specific commit

You can find a specific commit using the following command:

Terminal window

git

reflog

<commit-hash>

Recover lost commits or changes

If you accidentally deleted a branch or made changes that are no longer visible in the commit history, you can often recover them using the reflog. First, find the reference to the commit where the branch or changes existed, and then reset your branch to that reference.

Terminal window

git

reflog

<commit-hash>

git

reset

--hard

<commit-hash>

or you can use

HEAD@{n}

to reset to the nth commit before the one you want to reset to.

Terminal window

git

reflog

<commit-hash>

git

reset

--hard

HEAD@{

1

}

Conclusion

In this guide, we?ve covered important aspects of managing Git history through rebase and reflog. We?ve learned how rebase can help maintain a cleaner, more linear project history, and how reflog can help recover lost commits or changes.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Diff, Stash, Tags

Next

Collaborate with Github

Contribute

Community

Sponsor

URL: https://docs.chaicode.com/youtube/chai-aur-git/github/

About Github | Chai aur Docs

Skip to content

Chai aur Docs

Search

Ctrl

K

Cancel

YouTube

Instagram

LinkedIn

GitHub

X

Getting Started

Chai aur HTML

Welcome

HTML Intro

Emmet Crash Course

Common HTML Tags

Chai aur Git

Welcome

Git and GitHub

Terminology

Behind the scenes

Branches in Git

Diff, Stash, Tags

Managing History

Collaborate with Github

Chai aur C++

Welcome

C++ Intro

First Program in C++

Variables & Constants

Data Types

Operators

Control Flow

Loops

Functions

Chai aur Django

Welcome

Django Intro

Jinja Templates App

Tailwind Integration

Models

Relationships & Forms

Chai aur SQL

Welcome

SQL Intro

PostgreSQL

Database Design

Exercise - DB Design

SQL Joins and Keys

Exercise - Joins

Chai aur DevOps

Welcome

Server Startup

Nginx Configuration

Nginx Rate Limit

Nginx SSL Setup

Deploy Node API

PostgreSQL & Docker

On this page

Overview

What is Github?

Github Account

Configuring Git

Setup SSH Key

Publish Code to Remote Repository

Remote URL Setting

Add Remote Repository

Pushing Code

Setup an upstream remote

Get code from remote repository

Fetch code

Pull code

Conclusion

About Github

This guide will help you get started with Github.

What is Github?

Github is a web-based Git repository hosting service. It is a popular platform for developers to collaborate on projects and to share code. Github provides a user-friendly interface for managing and tracking changes to your code, as well as a platform for hosting and sharing your projects with others.

Some other alternative of Github are:

Gitlab

Bitbucket

Azure Repos

Gitea

But mainstream popular tool these days is Github.

Github Account

Creating a Github account is free and easy. You can create an account by visiting the

Github website

and clicking on the ?Sign up? button. You will be prompted to enter your email address and password, and then you will be redirected to the Github homepage.

Once you have created an account, you can start using Github to host and collaborate on your projects. Github provides a variety of features and tools that make it easy to manage and track your code, including issues, pull requests, and code reviews.

Configuring Git

If you haven?t done it already, you need to configure your git config file. You can do this by running the following command:

Terminal window

git

config

--global

user.email

"

"

git

config

--global

user.name

"

Your Name

"

This will set your email and name as your global settings. You can change these settings at any time by running the same command again.

Now you can check your config settings:

Terminal window

git

config

--list

This will show you all the settings that you have changed.

Setup SSH Key

If you haven?t done it already, you need to setup ssh key and add it to your github account. You can do this by following the instructions on the

Github website

.

You can find the exact steps on the website for both Windows and MacOS. The steps are same for both, only apple users need to add the ssh key to their keychain.

Generate a new SSH key

To generate a new SSH key, open the terminal and run the following command:

Terminal window

ssh-keygen

-t

ed25519

-C

"

"

Here ed25519 is the type of key that you are generating. This creates a new SSH key, using the provided email as label.

Save the key

After generating the key, you need to save it to your computer. You can do this by running the following command:

Enter a file in which to save the key

(/Users/YOU/.ssh/id_ALGORITHM): [Press enter]

At the prompt you can enter passphrase for the key or you can leave it blank. If you leave it blank, the key will

be saved without a passphrase.

Add key to your ssh-agent

After saving the key, you need to add it to your ssh-agent. You can do this by running the following command:

Here it is best to refer above link for more information, as Github has a lot of information on this. There is no point in repeating it here.

Add key to github

Use the web ui to add the key to your github account. You can do this by following the instructions on the Github website

.

Publish Code to Remote Repository

Now that you have setup your ssh key and added it to your github account, you can start pushing your code to the remote repository.

Create a new Repo on your system first, add some code and commit it.

Terminal window

git

init

git

add

<files>

git

commit

-m

"

commit message

"

Remote URL Setting

You can check the remote url setting by running the following command:

Terminal window

git

remote

-v

This will show you the remote url of your repository.

Add Remote Repository

You can add a remote repository by running the following command:

git remote add origin <remote-url>

Here

<remote-url>

is the url of the remote repository that you want to add and origin is the name of the remote repository. This

origin is used to refer to the remote repository in the future.

Terminal window

git

remote

add

origin

https://github.com/hiteshchoudhary/chai-something.git

Pushing Code

git push remote-name branch-name

Here

remote-name

is the name of the remote repository that you want to push to and

branch-name

is the name of the branch that you want to push.

Terminal window

git

push

origin

main

Setup an upstream remote

Setting up an upstream remote is useful when you want to keep your local repository up to date with the remote repository. It allows you to fetch and merge changes from the remote repository into your local repository.

To set up an upstream remote, you can use the following command:

Terminal window

git

remote

add

upstream

<remote-url>

or you can use shorthand:

Terminal window

git

remote

add

-u

<remote-url>

You can do this at the time of pushing your code to the remote repository.

Terminal window

git

push

-u

origin

main

This will set up an upstream remote and push your code to the remote repository.

This will allow you to run future commands like

git pull

and

git push

without specifying the remote name.

Get code from remote repository

There are two ways to get code from a remote repository:

fetch the code

pull the code

Fetch the code means that you are going to download the code from the remote repository to your local repository. Pull the code means that you are going to download the code from the remote repository and merge it with your local repository.

Fetch code

To fetch code from a remote repository, you can use the following command:

Terminal window

git

fetch

<remote-name>

Here

<remote-name>

is the name of the remote repository that you want to fetch from.

Pull code

To pull code from a remote repository, you can use the following command:

Terminal window

git pull <remote-name> <branch-name>

git

pull

origin

main

Here

<remote-name>

is the name of the remote repository that you want to pull from and

<branch-name>

is the name of the branch that you want to pull.

Conclusion

In this section, we have learned about Github and how to use it. We have also learned about how to setup ssh key and add it to your github account. We have also learned about how to publish code to the remote repository.

Start your journey with ChaiCode

All of our courses are available on

chaicode.com

. Feel free to check them out.

Complied by:

Hitesh Choudhary

Last updated:

Apr 14, 2025

Previous

Managing History

Next

Welcome

Contribute

Community

Sponsor