Welcome to our beginner-friendly guide on integrating GitHub with your Godot projects. This course will walk you through the basics of GitHub, its benefits, and how to get started. By the end of this guide, you will understand how to use GitHub to collaborate with others and back up your projects efficiently.

Introduction to GitHub

GitHub is a web-based Git repository used by millions of developers worldwide for programming, game development, and various technical projects. It serves as a powerful tool for backing up your projects and facilitating collaborative work.

Benefits of Using GitHub

- **Collaboration:** GitHub allows multiple people to work on the same project simultaneously, tracking changes and syncing updates effortlessly.
- **Backup:** By tracking changes and uploading them to its server, GitHub ensures that your project is backed up and accessible from any computer.
- **Efficiency:** Unlike transferring entire projects or individual files via USB or cloud storage, GitHub only updates specific changes, making the process more efficient.

Prerequisites

Before diving into the course, it is recommended that you have a basic understanding of Godot. However, this guide will focus primarily on GitHub Desktop, so prior knowledge of Godot is not strictly necessary.

Who is This Course For?

This course is ideal for:

- Individuals who want to collaborate on a project with multiple people.
- Those seeking a reliable way to back up their projects.

About Zenva

Zenva is an online learning academy with over 1 million students. We offer a wide range of courses suitable for beginners and those looking to learn something new. Our courses are versatile, allowing you to learn at your own pace and in your preferred style.

Let’s get started with the first lesson.

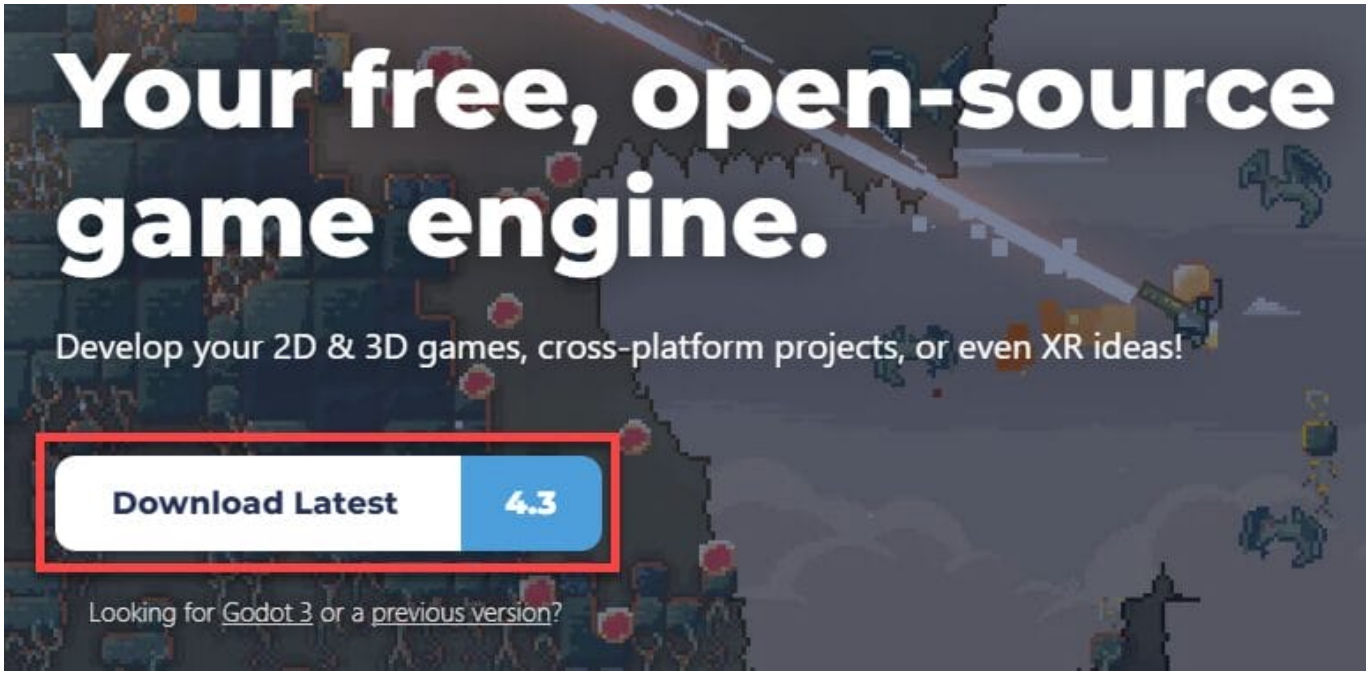
Which version of Godot should you use for this course?

Technology changes rapidly, so to make sure you have an optimal learning experience we recommend using **Godot 4.3** for this course – the latest stable release.

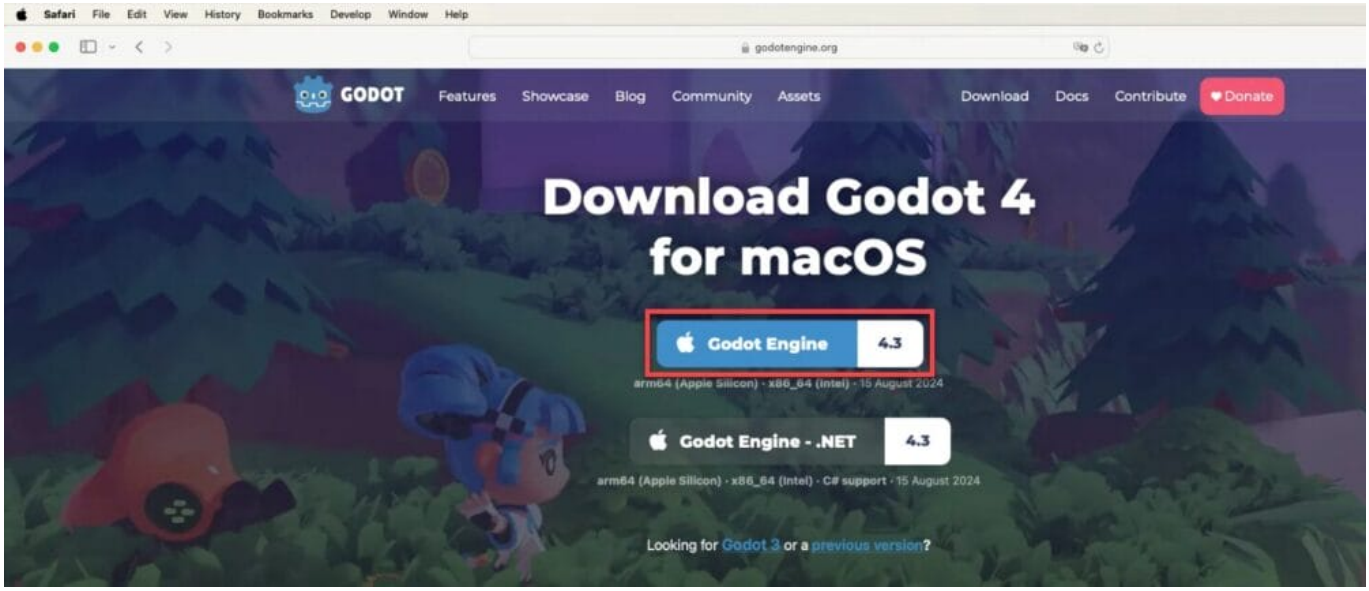
Please make sure not to use newer versions of the software than what we recommend, as the course material provided might not work as expected.

How to Install Version 4.3

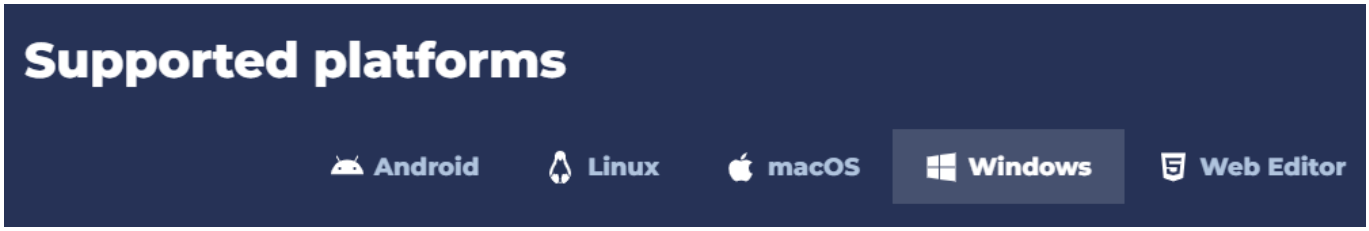
You can download the most recent version of Godot by heading to the Godot website (https://godotengine.org/) and clicking the “Download Latest” button on the front page. This will automatically take you to the download page for your operating system.



Once on the download page, just click the option for **Godot 4.3.** This will download the Godot engine to your local computer. From there, unzip the file and simply click on the application launcher – no further installation steps are required!



If Godot fails to automatically select the correct operating system for you, you can scroll down to the “Supported platforms” section on the download page to manually select the download version you would like to install:



Welcome to this introductory lesson on GitHub. In this article, we will explore the fundamentals of GitHub and its associated version control system, Git. Understanding these tools is crucial for collaborative projects, such as those you might undertake while learning at Zenva. By the end of this article, you will have a clear understanding of key concepts and be ready to set up your own repository.

Understanding Git and GitHub

GitHub is composed of two main systems:

- **Git**: An open-source version control system that tracks changes in files. For instance, if you modify your Godot project, Git can recognize which scripts, textures, or other assets have been altered, added, or removed.
- **GitHub**: A web-based Git repository hosting service. It simplifies the process by eliminating the need to install Git locally or set up a custom server. With GitHub, you can access your project files from anywhere with an internet connection.

Key Concepts in GitHub

To effectively use GitHub, you should be familiar with the following terms:

1. **Repository (Repo)**: This is where your project files are stored. Think of it as the folder or server where your scripts, textures, scenes, and 3D models reside.
2. **Collaborator**: These are the individuals contributing to the repository. Collaborators can make changes to the repo, and permissions can be set to control access levels.
3. **Commit**: When you make changes to your project, you commit those changes to the repo. This records all additions, removals, and modifications made to the files.
4. **Push**: This action applies your committed changes to the repo, making them available to all collaborators.
5. **Pull**: This action updates your local project with the most recent changes made to the repo by other collaborators.

How GitHub Works

The workflow in GitHub involves the following steps:

1. Make changes to your local project.
2. Commit those changes, recording what has been added, removed, or modified.
3. Push the committed changes to the GitHub server, updating the repo.
4. Pull any new changes made by other collaborators to update your local project.

This cycle ensures that everyone working on the project has the most up-to-date version, avoiding conflicts and ensuring smooth collaboration.

Benefits of Version Control

- Ensures all collaborators are working on the same version of the project.
- Provides a backup of your project, stored on GitHub’s servers.
- Facilitates easy access to your project from anywhere with an internet connection.

Setting Up GitHub

Now that you understand the basics of GitHub, the next step is to set it up and create your first repository. Stay tuned for our next article, where we will guide you through the process of setting up GitHub and creating a repo for your Godot project.

GitHub is a powerful tool that can greatly enhance your workflow, whether you are working solo or in a team. By using GitHub, you ensure that your project is safe, accessible, and collaborative.

Welcome to this beginner-friendly guide on how to sign up for GitHub and create a new repository for your Godot project. This article will walk you through the process step-by-step, ensuring you have a smooth experience setting up your version control system.

Signing Up for GitHub

1. Navigate to GitHub.com.
2. On the landing page, click the **Sign Up** button located at the top right corner.
3. Follow the prompts to create a new GitHub account. You will need to provide a username, email address, and create a password.
4. Once signed up, proceed to the sign-in page and log in with your newly created credentials.

Creating a New Repository

After logging in, you will be directed to your dashboard. GitHub offers numerous features, but for this guide, we will focus on creating a repository for your Godot project.

Understanding Repositories

- **Public Repositories:** Accessible to anyone on the internet.
- **Private Repositories:** Visible only to you and any collaborators you invite.

Repositories can serve as a portfolio to showcase your projects publicly. Additionally, GitHub can host websites directly from your repositories.

Steps to Create a New Repository

1. On the left-hand side of your dashboard, click the **New** button or navigate to your profile icon at the top right and select **Your repositories**.
2. Click the green **New** button located at the top left corner to create a new repository.
3. Fill in the repository details:
    - **Owner:** Ensure this is your account.
    - **Repository Name:** Provide a unique name, e.g., “My First Godot Project”.
    - **Description:** Optional, but can be useful for context.
    - **Public/Private:** Choose based on your preference for visibility.
4. Add a **.gitignore** file:
    - Select the **Add .gitignore** option.
    - Search for and select the **Godot** template. This file helps exclude unnecessary metadata files from your repository.
5. Click **Create Repository** to finalize.

Exploring Your New Repository

After creation, you will be directed to your repository’s main page. Here’s what you can do:

- View your project name and list of files.
- Navigate through files and folders similar to a file explorer.
- View the **.gitignore** file contents, which list excluded file extensions and directories.
- Track commits made by collaborators, showing added, removed, or modified files line by line.

Next Steps

In the next lesson, we will explore how to connect your Godot project to this repository using GitHub Desktop. Stay tuned for more detailed instructions on managing your project with version control.

Thank you for following along. See you in the next lesson!

Welcome to this beginner-friendly guide on setting up and installing GitHub Desktop. In this lesson, we will walk through the process of connecting your GitHub repository to your local machine using GitHub Desktop, allowing you to track and manage changes to your projects efficiently.

Introduction to GitHub Desktop

GitHub Desktop is a user-friendly application that enables you to manage your GitHub repositories directly from your computer. It allows you to detect changes made to your project files and commit those changes to your GitHub repository.

Why Use GitHub Desktop?

- Easily track changes made to your project files.
- Commit and push changes to your GitHub repository.
- Simplifies the process of managing your repositories.

Downloading GitHub Desktop

1. Visit desktop.github.com or search for “GitHub Desktop” in your web browser.
2. Click on the download button to start the download process.

Connecting GitHub Desktop to Your GitHub Account

1. Open GitHub Desktop after installation.
2. Go to **File** > **Options** > **Accounts**.
3. Click on **Sign in to GitHub.com**.
4. Select **Continue with browser** and log in to your GitHub account.
5. Verify that you are logged in by checking **File** > **Options**.

Configuring Your Git Settings

1. In the **Options** menu, go to the **Git** section.
2. Enter the name and email you want to use for committing changes.
3. Click **Save** to apply the settings.

Cloning a Repository

1. Go to **File** > **Clone Repository**.
2. Select the repository you want to clone from your GitHub account.
3. Choose a local path to store the repository (e.g., create a folder named “GitHub” on your computer).
4. Click **Clone** to download the repository to your local machine.

Managing Changes in GitHub Desktop

Once your repository is cloned, you can manage changes directly from GitHub Desktop:

- View the current repository and any changes made.
- Check the history of commits and pushes.
- Commit and push changes to your GitHub repository.

Making and Committing Changes

1. Make changes to your project files in the local directory.
2. In GitHub Desktop, view the changes listed under the **Changes** tab.

3. Enter a commit message and an optional description.
4. Click **Commit to main** to commit the changes locally.
5. Click **Push origin** to push the changes to your GitHub repository.

Viewing Commit History

To view the history of commits:

1. Go to the **History** tab in GitHub Desktop.
2. See a list of all commits and their details.

Thank you for following this guide. You are now equipped with the knowledge to set up and use GitHub Desktop to manage your projects effectively. Happy coding!

Welcome to another insightful lesson at Zenva! In this tutorial, we will guide you through the process of adding your Godot project to a GitHub repository. Whether you have an existing Godot project or plan to create a new one, this article will help you understand the steps involved in integrating your project with GitHub.

Prerequisites

- Basic understanding of Godot and GitHub.
- A GitHub account and GitHub Desktop installed.
- A Godot project (existing or new).

Locating or Creating Your Godot Project

First, locate your Godot project or create a new one. Open the project folder in your File Explorer. You should see a structure similar to the following:

- .gitattributes
- .gitignore
- assets/
- project.godot

Understanding GitHub Repository Structure

On the left side of your screen, you should see your GitHub repository. This directory will store all the files and changes you want to upload to your GitHub repository. It typically includes:

- .gitignore
- Example files or directories you have created.

Managing .gitignore and .gitattributes Files

Godot projects come with built-in .gitignore and .gitattributes files. However, the default .gitignore file is quite basic. It is recommended to use the more comprehensive .gitignore file provided by GitHub. Here’s what you need to do:

1. Delete the existing .gitignore file in your Godot project.
2. Keep the default .gitattributes file as it is.

Adding Your Godot Project to the Repository

To add your Godot project to the GitHub repository, follow these steps:

1. Select all files and folders in your Godot project.
2. Drag and drop them into your GitHub repository directory.

Your GitHub repository will now act as your project folder. To edit the project, open Godot and select the repository folder.

Committing and Pushing Changes

Once you have added your files to the repository, you need to commit and push these changes. Here’s how:

1. Open GitHub Desktop and navigate to your repository.
2. You will see a list of files that have been added or changed.

3. Provide a commit message, such as “Added Godot project,” and click “Commit to main.”
4. Click “Push origin” to upload your changes to the GitHub server.

Verifying the Upload

To ensure your files have been successfully uploaded:

1. Go to the “History” tab in GitHub Desktop to see the commit history.
2. Refresh your GitHub repository on the website to verify that all files and folders are present.

Opening the Project in Godot

To open your project from the GitHub repository in Godot:

1. Open Godot and click the “Import” button.
2. Navigate to your GitHub repository folder and select it.
3. Click “Import and Edit” to open the project.

Any changes made to the project will now appear in the change list in GitHub Desktop.

Next Steps

In the next lesson, we will explore how to make changes to your project, push them to GitHub, and handle merge conflicts when collaborating with others. Stay tuned for more!

Thank you for reading, and we look forward to seeing you in the next lesson!

Welcome back! In this lesson, we will explore how to make changes to our Godot project and review those changes before committing them. Specifically, we will cover creating new files, modifying existing files, and deleting files. Let's dive right in!

Creating a New File

To begin, let's create a new script file in our Godot project:

1. Right-click in the FileSystem dock.
2. Select New Script.
3. Name the script test_script.gd.

Your new script will appear with the default code:

```
extends Node

func _ready():
    pass # Replace with function body.

func _process(delta):
    pass # Replace with function body.
```

Modifying an Existing File

Next, let's modify an existing script. For this example, we'll add a print statement to one of our scripts:

1. Open an existing script, for instance, player.gd.
2. Add the following line of code:

```
print("Add gravity")
```

This line will print "Add gravity" to the output when the script runs.

Deleting a Line of Code

Now, let's delete a line of code from another script:

1. Open a different script, such as enemy.gd.
2. Delete a function or a line of code. For example, remove the following function:

```
func _ready() -> void:
    change_state("idle")
```

Deleting a File

Finally, let's delete a file to see how it reflects in GitHub Desktop:

1. Navigate to the assets folder.
2. Delete a file, for example, a random tile image tile.png.

Reviewing Changes in GitHub Desktop

After making these changes, open GitHub Desktop to review them:

- **New Files:** Files with a green icon indicate they are new.
- **Modified Files:** Files with a yellow icon have been modified.
- **Deleted Files:** Files with a red icon have been deleted.

You can click on each file to see the specific changes made:

- New lines of code are highlighted with a plus icon.
- Deleted lines of code are highlighted in red.

Committing and Pushing Changes

Once you have reviewed your changes, you can commit and push them to the repository:

1. Enter a commit message, such as Testing out GitHub.
2. Click Commit to main.
3. Click Push origin to push the changes to the remote repository.

Viewing Commit History

To view the history of your commits:

1. Go to the History tab in GitHub Desktop.
2. Click on a commit to see the details of the changes made.

This history allows you to track changes and revert to previous versions if needed.

Challenge

As a challenge, set up your Godot project in GitHub (if you haven’t already) and begin creating commits by changing files and pushing those changes to the repository.

That’s it for this lesson! In the next lesson, we will continue exploring more features of Godot and GitHub integration.

In this lesson, we will explore how to add collaborators to your GitHub project. A collaborator is someone who can contribute to and access your repository. By default, when you create a new repository, you are the sole collaborator. If you are working alone and using GitHub for storage and backup, you do not need to add collaborators. However, if you are working on a project with a team, adding collaborators is essential to allow them to view, download, and make changes to the project.

Steps to Add Collaborators

1. **Create a GitHub Account:** Ensure that the people you want to add as collaborators have their own GitHub accounts.
2. **Navigate to Your Repository:** Go to the GitHub repository where you want to add collaborators.
3. **Access Settings:** At the top of the repository page, click on the **Settings** button.
4. **Select Collaborators:** On the left-hand side, under the **Access** section, select **Collaborators**.
5. **Add People:**
    - Click on the **Add people** button.
    - Search for the collaborator's account using their email address or username.
    - Click the green **Add** button to send them an invitation email.
6. **Accept Invitation:** The collaborator will receive an email invitation. Once they accept, the repository will appear in their list, and they will have full access to view, connect to GitHub Desktop, push changes, and pull changes.

Benefits of Adding Collaborators

- **Teamwork:** Multiple people can work on the same GitHub repository, fostering collaboration and teamwork.
- **Efficient Project Management:** Collaborators can contribute to the project, making it easier to manage and complete tasks efficiently.
- **Version Control:** All collaborators can push and pull changes, ensuring that the project is up-to-date and well-maintained.

By following these steps, you can easily add collaborators to your GitHub project and enhance your team's productivity and collaboration.

In this lesson, we will explore how to pull changes from a GitHub repository using GitHub Desktop. This process is essential for keeping your local project up-to-date with changes made by other collaborators. We will simulate another user making changes to the repository and then update our local version by pulling those changes.

Simulating Changes in the Repository

Since we don't have another user or computer to simulate this process, we will make changes directly on the GitHub website. Here's how you can do it:

1. Navigate to your GitHub repository on the GitHub website.
2. Open the scripts directory and select one of the GDScript files, such as GameOver.gd.
3. Click the Edit button at the top right corner of the file.
4. Add a comment or any change to the file. For example, add the following comment:

```
# Added for pull test
```

5. Commit the changes with a message. For instance, you can write Updated GameOver.gd as the commit message.
6. Click Commit changes to save the changes directly to the repository.

Pulling Changes in GitHub Desktop

Now that we have made changes to the repository, let's pull those changes into our local project using GitHub Desktop:

1. Open GitHub Desktop and navigate to your repository.
2. Click the Fetch origin button at the top right corner. This action will check the repository for any new commits.
3. Once fetched, you will see a Pull origin button indicating that there are new commits to pull.
4. Click Pull origin to pull the changes into your local project.

Verifying the Changes

After pulling the changes, you can verify them in your local project:

1. Open the History tab in GitHub Desktop to see the list of commits. You should see the new commit Updated GameOver.gd.
2. Open your Godot project. You might see a message indicating that some files are newer on disk.
3. Click Discard local changes and reload to apply the new changes.
4. Navigate to the GameOver.gd script in your Godot project to see the new line of code added:

```
# Added for pull test
```

Best Practices

- **Close Godot Before Pulling Changes:** It is generally a good practice to close the Godot engine before pulling changes to avoid conflicts and ensure that all files are updated correctly.
- **Commit Local Changes Before Pulling:** If you have local changes, commit them before pulling from the repository to prevent conflicts and overwriting files.

In the next lesson, we will cover an important topic: merge conflicts. This is a common issue when

collaborating with others, and understanding how to handle merge conflicts is crucial for smooth collaboration.

Thank you for following along, and see you in the next lesson!

In this lesson, we will explore the concept of merge conflicts in Git. Merge conflicts occur when changes made to a repository (repo) have not been pulled to your local machine, and you make changes on your local machine that interfere with the changes in the repo. This situation arises when multiple people work on the same file simultaneously without proper communication.

Understanding Merge Conflicts

A merge conflict happens when:

- Changes are made to the repo that you have not pulled to your local machine.
- You make changes on your local machine that directly interfere with the changes in the repo.
- You try to pull the changes from the repo to commit your changes, realizing that you have been working on the same file as someone else at the same time.

Simulating a Merge Conflict

To understand how to resolve merge conflicts, we will simulate one. Follow these steps:

1. Navigate to the repo on the GitHub website.
2. Go to the scripts folder, then to monsters, and select the alien file.
3. Edit the file to modify the same line in both the repo and your local machine.

For example, let’s modify the gravity application in the alien file:

```
// Original code
velocity.y += gravity * delta

// Modified code in the repo
velocity.y += gravity * delta * 2
```

Commit this change to the repo. Now, let’s simulate working on the same file locally without pulling the changes from the repo.

Making Local Changes

Open the same file in your local editor and make a different change:

```
// Original code
velocity.y += gravity * delta

// Modified code locally
var gravity_mod : float = 3.5
velocity.y += gravity * delta * gravity_mod
```

Commit this change locally. Now, try to pull the changes from the repo.

Resolving the Merge Conflict

When you try to pull the changes, GitHub Desktop will indicate a conflict because the changes are on

the same lines of code. Follow these steps to resolve the conflict:

1. Click on Fetch origin to retrieve the changes from the repo.
2. Click on Pull origin to pull the changes, which will indicate a conflict.
3. Click on Open in Visual Studio Code to resolve the conflict.

In Visual Studio Code, you will see the conflict with the following options:

- **Accept Current Change:** Override the repo's changes with your local changes.
- **Accept Incoming Change:** Replace your local changes with the repo's changes.
- **Accept Both Changes:** Incorporate both changes and modify as needed.
- **Compare Changes:** Compare the changes side by side.

For this example, we'll accept the current change. Save the file and exit Visual Studio Code. GitHub Desktop will indicate that all conflicts have been resolved. Click on Continue merge to create a new commit that merges the changes.

Best Practices

To avoid merge conflicts:

- Communicate with your team about who is modifying which files and when.
- Regularly pull changes from the repo to keep your local machine up to date.

By following these steps, you can effectively resolve merge conflicts and ensure smooth collaboration in your Git workflow.

Congratulations on completing the course! You have just taken a significant step in your journey to mastering game development with Godot and version control with GitHub. This article will summarize the key points covered in the video transcript, ensuring you have a clear understanding of the concepts and tools discussed.

Introduction to GitHub and Godot

In this course, we explored the integration of GitHub with Godot. GitHub is a web-based Git repository hosting service that allows you to manage and track changes to your code. This is particularly useful for collaborative projects and for backing up your work.

Setting Up Your GitHub Repository

To get started, you need to:

1. Create a GitHub account.
2. Set up a new repository (repo) on GitHub.
3. Download and install GitHub Desktop, which enables you to interact with your GitHub repository on your local machine.

Working with Godot Projects

Once your repository is set up, you can:

1. Clone your repo to your local machine.
2. Set up your Godot project within the cloned repository.
3. Sync your project files with the GitHub repository.

Key GitHub Terminology

Understanding the following terms will help you navigate GitHub more effectively:

- **Push:** Uploading your local changes to the remote repository.
- **Pull:** Downloading changes from the remote repository to your local machine.
- **Commit:** A snapshot of your repository at a specific point in time.
- **Repository:** A storage space for your project files and their revision history.
- **Collaborator:** Someone who has been given access to contribute to your repository.
- **Merge Conflict:** A situation where changes from different branches cannot be automatically merged.

Benefits of Using GitHub

There are two primary reasons for using GitHub with your Godot projects:

1. **Collaboration:** GitHub is ideal for projects involving multiple people. It allows you to update and sync small changes with others, which is more efficient than using methods like Google Drive or USB sticks. GitHub handles merge conflicts and provides a user-friendly interface.
2. **Backup:** Connecting your Godot project to a GitHub repository ensures that your files are backed up. This protects your work from accidental loss or corruption.

About Zenva

Zenva is an online learning academy with over 1 million students. We offer a wide range of courses suitable for beginners and those looking to learn something new. Our courses are versatile, allowing you to learn through:

- Online video tutorials
- Lesson summaries
- Following along with the instructor using included course files

Thank you for following along with the course. We wish you the best of luck with your future Godot games!