

GitHub

GitHub is a web-based hosting service for version control using git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

GitHub offers plans for both private repositories and free accounts which are commonly used to host open-source software projects.

Note: visit <https://help.github.com> for detailed information.

Features

Code Review

Propose changes

Better code starts with a Pull Request, a living conversation about changes where you can talk through ideas, assign tasks, discuss details, and conduct reviews.

Request reviews

If you're on the other side of a review, you can request reviews from your peers to get the exact feedback you need.

See the difference

Reviews happen faster when you know exactly what's changed. Diffs compare versions of your source code side by side, highlighting the parts that are new, edited, or deleted.

Protect branches

Only merge the highest quality code. You can configure repositories to require status checks, reducing both human error and administrative overhead.

Project Management

[See your project's big picture](#)

See everything happening in your project and choose where to focus your team's efforts with Projects, task boards that live right where they belong: close to your code.

[Track and assign tasks](#)

Issues help you identify, assign, and keep track of tasks within your team. You can open an Issue to track a bug, discuss an idea with an @mention, or start distributing work.

Integration

Can be integrated with many development environments.

Team Management

Social Coding

[Follow projects](#)

Starring repositories lets maintainers know you appreciate their work and helps you track projects you don't contribute to. Watch repositories to get notifications when someone opens an issue or submits a pull request

[Explore your interests](#)

Get data powered project recommendations in your news feed. And with [Explore](#), you can browse curated [collections](#), trending repositories, and popular [topics](#).

[Share your achievements](#)


Show the public activity and proud moments behind your green squares. Pin your best work to your profile or browse others' timelines to see the projects they've shaped.

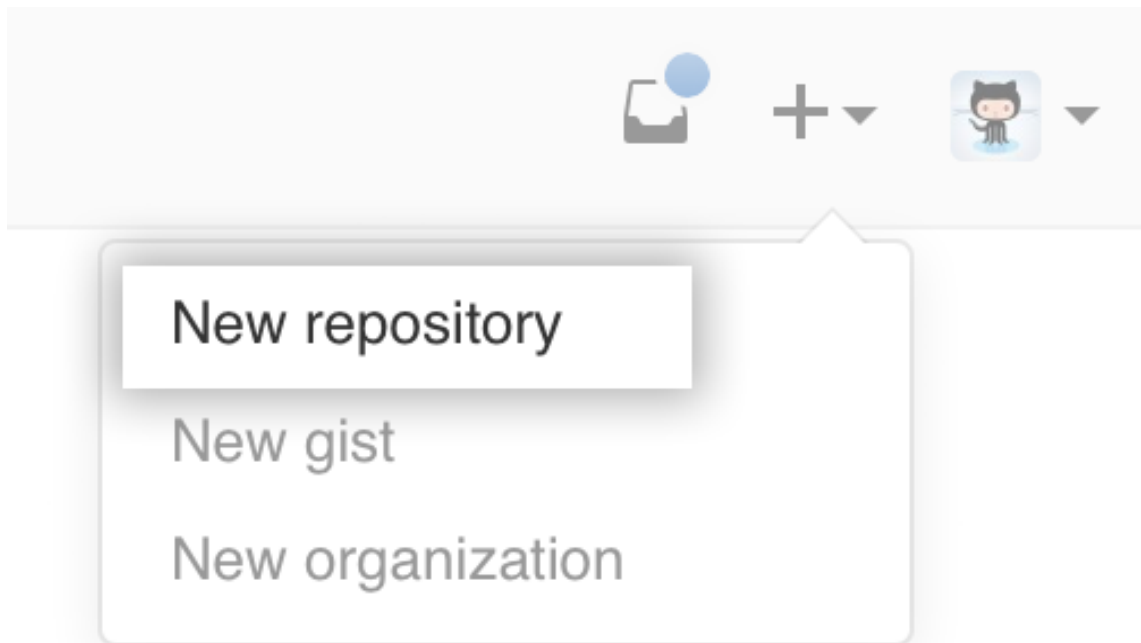
Documentation

Code Hosting

Creating a new repository

Step 1

In the upper-right corner of any page, click , and then click **New repository**.

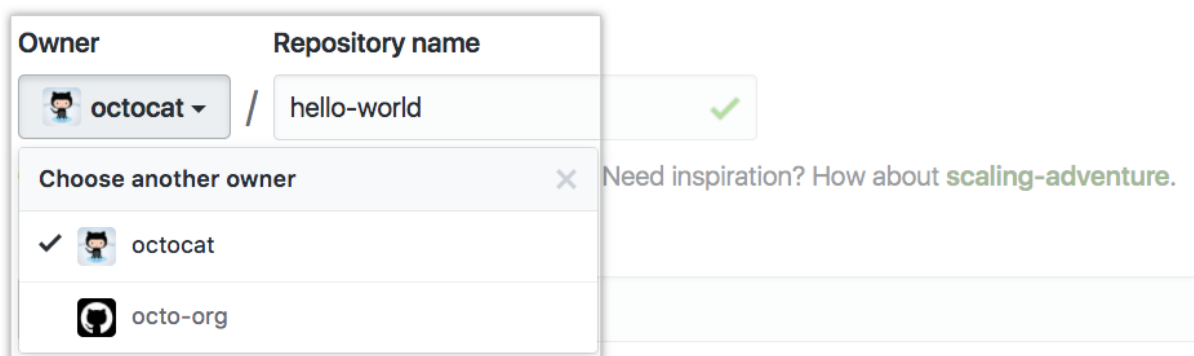


Step 2

In the Owner drop-down, select the account you wish to create the repository on.

Create a new repository

A repository contains all the files for your project, including the revision history.

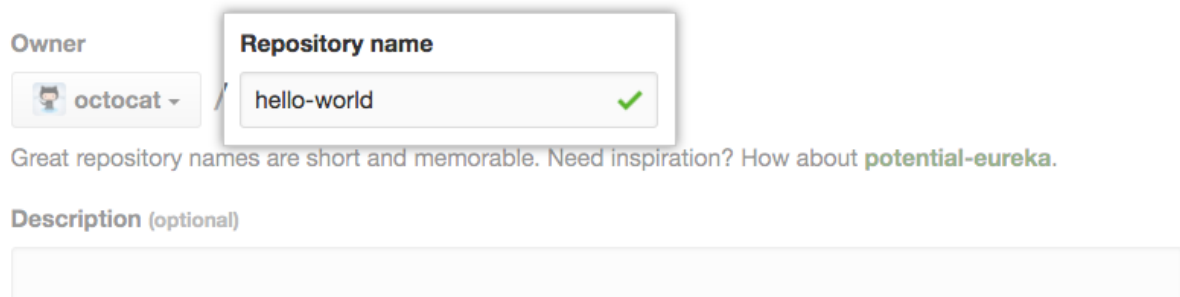
A screenshot of the 'Create a new repository' form. The form has two main sections: 'Owner' and 'Repository name'. The 'Owner' section has a dropdown menu with 'octocat' selected. Below it, there is a list of suggestions: 'octocat' (checked) and 'octo-org'. The 'Repository name' section has a text input field with 'hello-world' entered. To the right of the input field is a green checkmark icon. Below the form, there is a link that says 'Need inspiration? How about scaling-adventure.'

Step 3

Type a name for your repository, and an optional description.

Create a new repository

A repository contains all the files for your project, including the revision history.



Owner: octocat

Repository name: hello-world ✓

Great repository names are short and memorable. Need inspiration? How about [potential-eureka](#).

Description (optional):

Step 4

You can choose to make the repository either public or private. Public repositories are visible to the public, while private repositories are only accessible to you, and people you share them with. Your account must be on a [paid plan](#) to create a private repository.

Step 5

There are a number of optional items you can pre-populate your repository with. If you're importing an existing repository to GitHub, don't choose any of these options, as you may introduce a merge conflict. You can choose to add these files using the command line later.

- You can create a README, which is a document describing your project.
- You can create a CODEOWNERS file, which describes which individuals or teams own certain files in the repository.
- You can create a .gitignore file, which is a set of ignore rules.
- You can choose to add a software license for your project.

Step 6

When you're finished, click **Create repository**.

Step 7

At the bottom of the resulting Quick Setup page, under "Import code from an old repository", you can choose to import a project to your new repository. To do so, click **Import code**.

TortoiseGit

- TortoiseGit is a Windows Shell Interface to Git and based on TortoiseSVN. It's open source and can fully be build with freely available software.
- Since it's not an integration for a specific IDE like Visual Studio, Eclipse or others, you can use it with whatever development tools you like, and with any type of file. Main interaction with TortoiseGit will be using the context menu of the Windows explorer.
- TortoiseGit supports you by regular tasks, such as committing, showing logs, diffing two versions, creating branches and tags, creating patches and so on (see our Screenshots or documentation).

Features of TortoiseGit

- Easy to use
 - all commands are available directly from the Windows Explorer
 - only commands that make sense for the selected file/folder are shown. You won't see any commands that you can't use in your situation.
 - See the status of your files directly in the Windows explorer
 - descriptive dialogs, constantly improved due to user feedback
 - allows moving files by right-dragging them in the Windows explorer
- Powerful commit dialog
 - integrated spell checker for log messages
 - auto completion of paths and keywords of the modified files
 - text formatting with special chars

C# Structure

In C#, classes and structs are blueprints that are used to create instance of a class. Structs are used for lightweight objects such as Color, Rectangle, Point etc. Unlike class, structs in C# are value type than reference type. It is useful if you have data that is not intended to be modified after creation of struct.

The **struct** keyword is used for creating a structure.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member for your program.

For example, here is the way you can declare the Book structure

```
struct Books {  
    public string title;  
    public string author;  
    public string subject;  
    public int book_id;  
};
```

Let's see a simple example of struct Rectangle which has two data members width and height.

```
using System;
```

```
public struct Rectangle
{
    public int width, height;
}
public class TestStructs
{
    public static void Main()
    {
        Rectangle r = new Rectangle();
        r.width = 4;
        r.height = 5;
        Console.WriteLine("Area of Rectangle is: " + (r.width * r.height));
    }
}
```

Area of Rectangle is: 20

C# Struct Example: Using Constructor and Method

```
using System;
public struct Rectangle
{
    public int width, height;

    public Rectangle(int w, int h)
    {
        width = w;
        height = h;
    }
    public void areaOfRectangle() {
        Console.WriteLine("Area of Rectangle is: "+(width*height)); }
}
public class TestStructs
{
    public static void Main()
    {
        Rectangle r = new Rectangle(5, 6);
        r.areaOfRectangle();
    }
}
```

```
}
```

Area of Rectangle is: 30

Features of C# Structures

You have already used a simple structure named Books. Structures in C# are quite different from that in traditional C or C++. The C# structures have the following features –

- Structures can have methods, fields, indexers, properties, operator methods, and events.
- Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure. The default constructor is automatically defined and cannot be changed.
- Unlike classes, structures cannot inherit other structures or classes.
- Structures cannot be used as a base for other structures or classes.
- A structure can implement one or more interfaces.
- Structure members cannot be specified as abstract, virtual, or protected.
- When you create a struct object using the **New** operator, it gets created and the appropriate constructor is called. Unlike classes, structs can be instantiated without using the New operator.
- If the New operator is not used, the fields remain unassigned and the object cannot be used until all the fields are initialized.

Class versus Structure

Classes and Structures have the following basic differences –

- classes are reference types and structs are value types
- structures do not support inheritance
- structures cannot have default constructor

C# Enum

An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.

C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

Declaring *enum* Variable

The general syntax for declaring an enumeration is

```
enum <enum_name> {  
    enumeration list  
};
```

Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example –

```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
```

C# Enum Example

```
using System;  
public class EnumExample  
{  
    public enum Season { WINTER, SPRING, SUMMER, FALL }  
  
    public static void Main()  
    {  
        int x = (int)Season.WINTER;  
        int y = (int)Season.SUMMER;  
        Console.WriteLine("WINTER = {0}", x);  
        Console.WriteLine("SUMMER = {0}", y);  
    }  
}
```

```
WINTER = 0  
SUMMER = 2
```

C# enum example for Days

```
using System;

public class EnumExample
{
    public enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    public static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Mon;
        int z = (int)Days.Sat;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Mon = {0}", y);
        Console.WriteLine("Sat = {0}", z);
    }
}
```

```
Sun = 0
Mon = 1
Sat = 6
```

C# enum example: traversing all values using getNames()

```
using System;

public class EnumExample
{
    public enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    public static void Main()
    {

```

```
        foreach (string s in Enum.GetNames(typeof(Days)))
        {
            Console.WriteLine(s);
        }
    }
}
```

```
Sun
Mon
Tue
Wed
Thu
Fri
Sat
```

C# enum example: traversing all values using `getValues()`

```
using System;
public class EnumExample
{
    public enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    public static void Main()
    {
        foreach (Days d in Enum.GetValues(typeof(Days)))
        {
            Console.WriteLine(d);
        }
    }
}
```

```
Sun
Mon
Tue
Wed
Thu
Fri
Sat
```

References

1. <https://help.github.com>
2. <https://tortoisegit.org/about/>
3. https://www.tutorialspoint.com/csharp/csharp_struct.htm
4. <https://www.javatpoint.com/c-sharp-structs>
5. <https://www.javatpoint.com/c-sharp-enum>
6. https://www.tutorialspoint.com/csharp/csharp_enums.htm
7. <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/enum>
8. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-structs>