

Introducción

Discord es una aplicación de chat que permite a millones de usuarios de todo el mundo transmitir mensajes y voz en línea en comunidades llamadas gremios o servidores. Discord también proporciona una amplia API que los desarrolladores pueden usar para crear potentes bots de Discord. Los bots pueden realizar diversas acciones, como enviar mensajes a servidores, mensajes directos a usuarios, moderar servidores y reproducir audio en los chats de voz. Esto permite a los desarrolladores crear bots potentes que incluyen funciones avanzadas y complejas como herramientas de moderación o, incluso, juegos. Por ejemplo, el bot de utilidad Dyno sirve a millones de gremios y contiene características útiles como la protección contra spam, un reproductor de música y otras funciones útiles. Aprender a crear bots de Discord le permite implementar muchas posibilidades, con las que miles de personas podrían interactuar cada día.

En este tutorial, creará un bot de Discord desde cero, usando Node.js y la biblioteca de Discord.js, que permite a los usuarios interactuar directamente con la API de Discord. Configuraré un perfil para un bot de Discord, obtendrá tokens de autenticación para el bot y lo programaré con la capacidad de procesar comandos con argumentos, desde los usuarios.

Requisitos previos

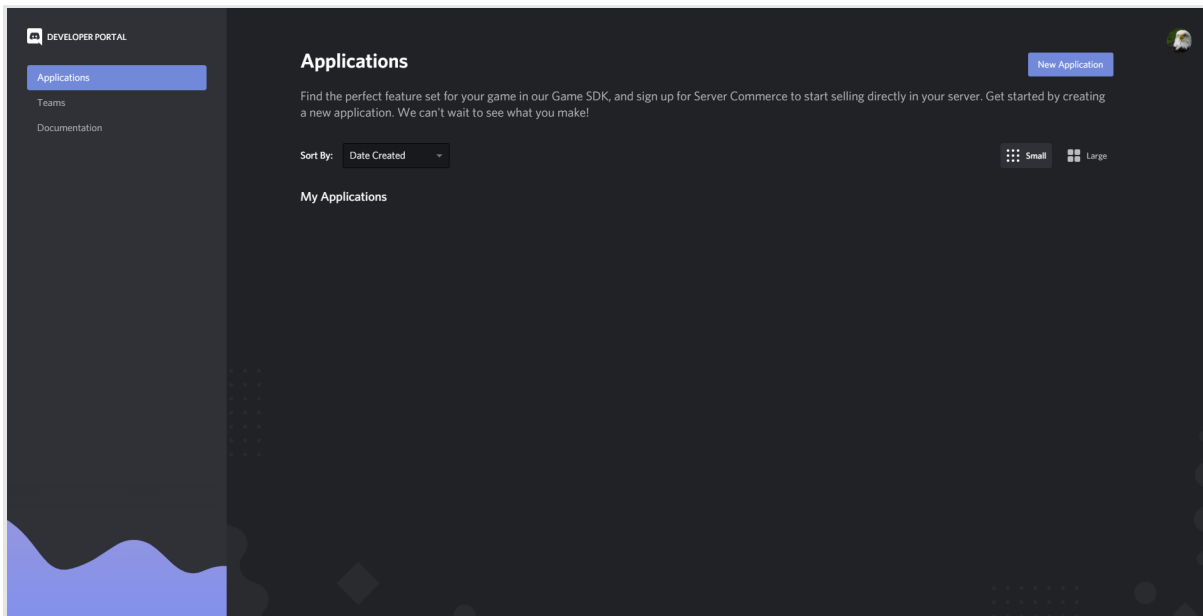
Para comenzar, necesitará lo siguiente:

- Node.js instalado en su equipo de desarrollo. Para instalarlo en macOS o Ubuntu 18.04, siga los pasos de [Cómo instalar Node.js y crear un entorno de desarrollo local en macOS](#) o las indicaciones de la sección [Instalación con un PPA](#), de **Cómo instalar Node.js en Ubuntu 18.04**.
- Cualquier editor de texto que elija, como Visual Studio Code, Atom, Sublime o Nano.
- Una cuenta de Discord gratuita con una cuenta de correo electrónico verificada y un servidor de Discord gratuito que utilizará para probar su bot de Discord.

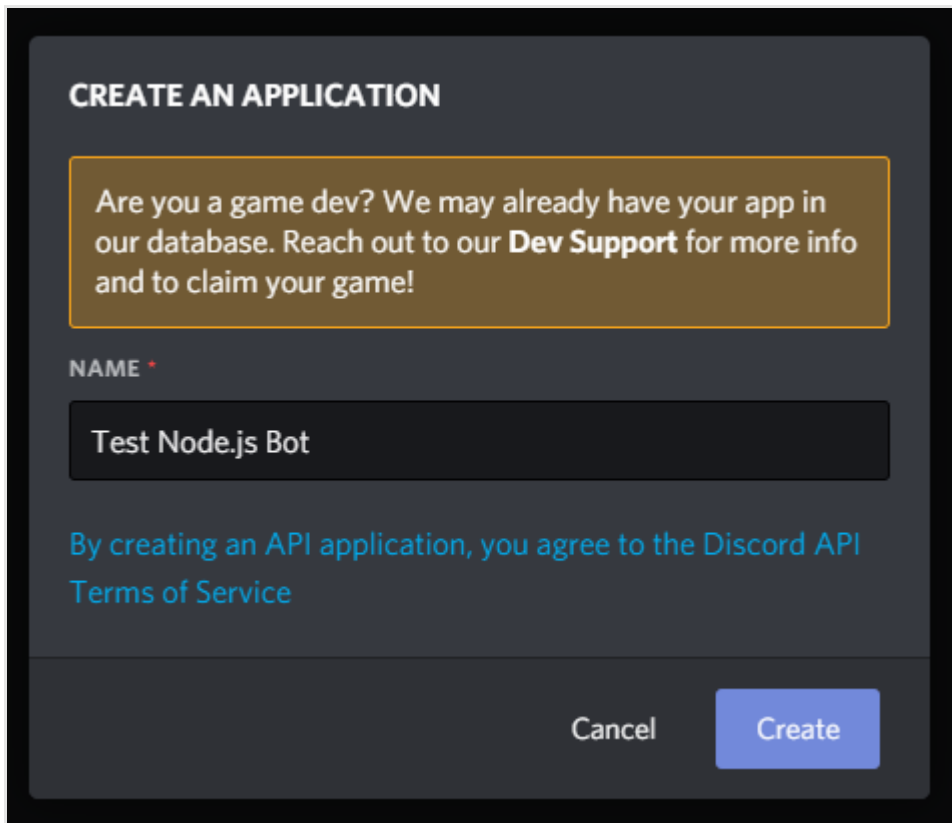
Paso 1: Configurar un bot de Discord

En este paso, utilizará la GUI de desarrolladores de Discord para configurar un bot de Discord y obtener el token de bot, que pasará a su programa.

Para registrar un bot en la plataforma Discord, utilice el [panel de aplicación de Discord](#). Aquí los desarrolladores pueden crear aplicaciones de Discord incluyendo bots de Discord.



Para comenzar, haga clic en **Nueva aplicación**. Discord le solicitará que introduzca un nombre para su nueva aplicación. A continuación, haga clic en **Crear** para crear la aplicación.



CREATE AN APPLICATION

Are you a game dev? We may already have your app in our database. Reach out to our **Dev Support** for more info and to claim your game!

NAME *

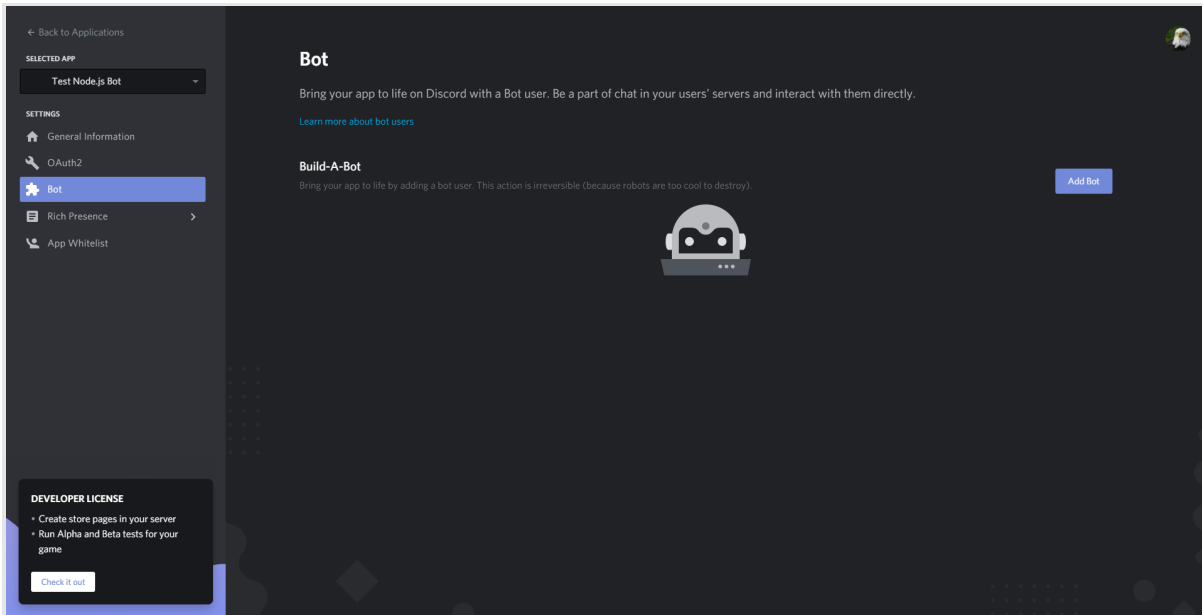
Test Node.js Bot

By creating an API application, you agree to the [Discord API Terms of Service](#)

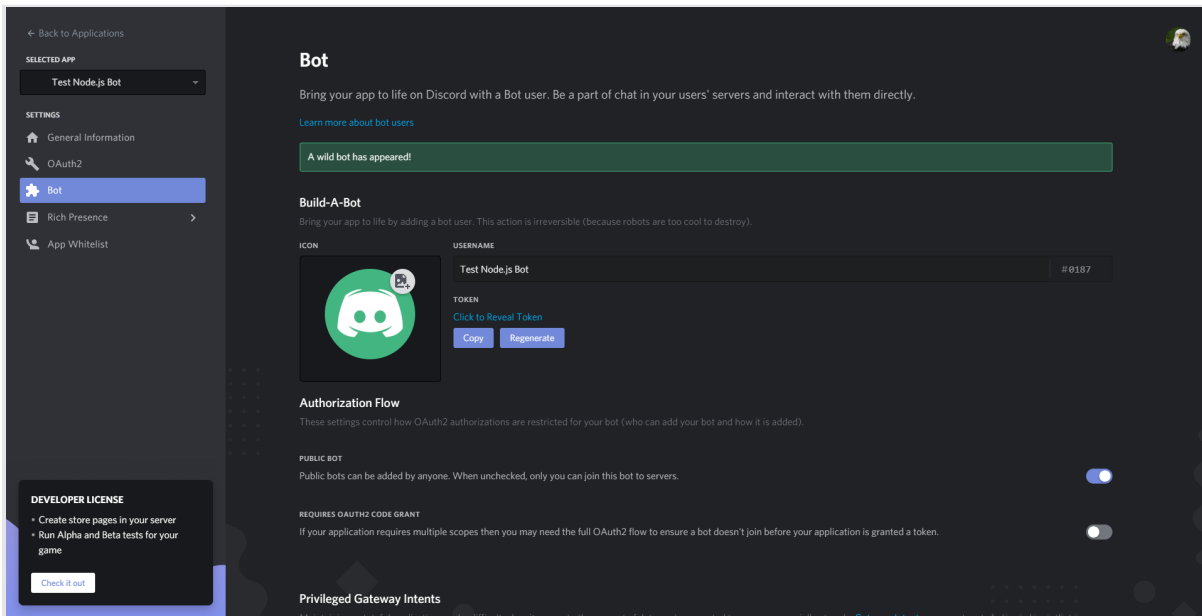
Cancel Create

Nota: El nombre de su aplicación es independiente del nombre del bot, y el bot no tiene el mismo nombre que la aplicación.

Ahora, abra el panel de su aplicación. Para agregar un bot a la aplicación, diríjase a la pestaña **Bot** en la barra de navegación a la izquierda.



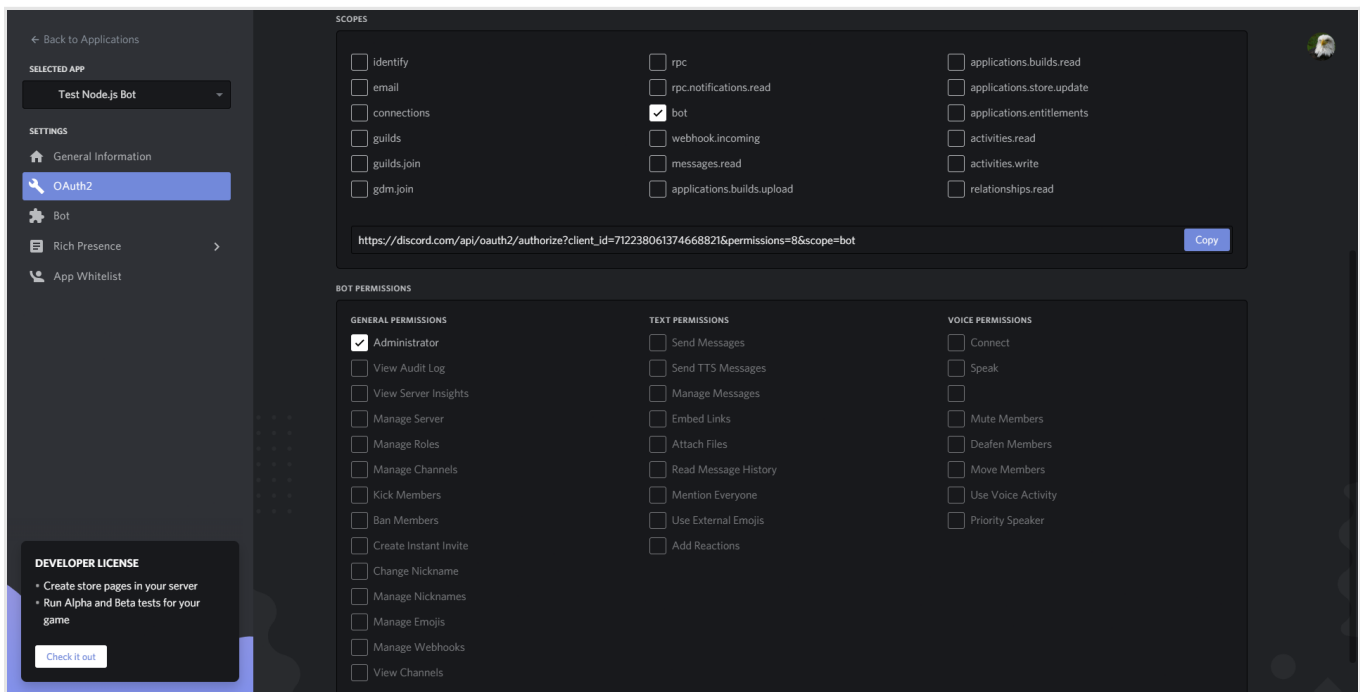
Haga clic en el botón **Añadir bot** para agregar un bot a la aplicación. Haga clic en el botón **Sí, ¡hazlo!** cuando le pida la confirmación. A continuación, estará en un panel que contiene detalles sobre el nombre de su bot, el token de autenticación y la imagen de perfil.



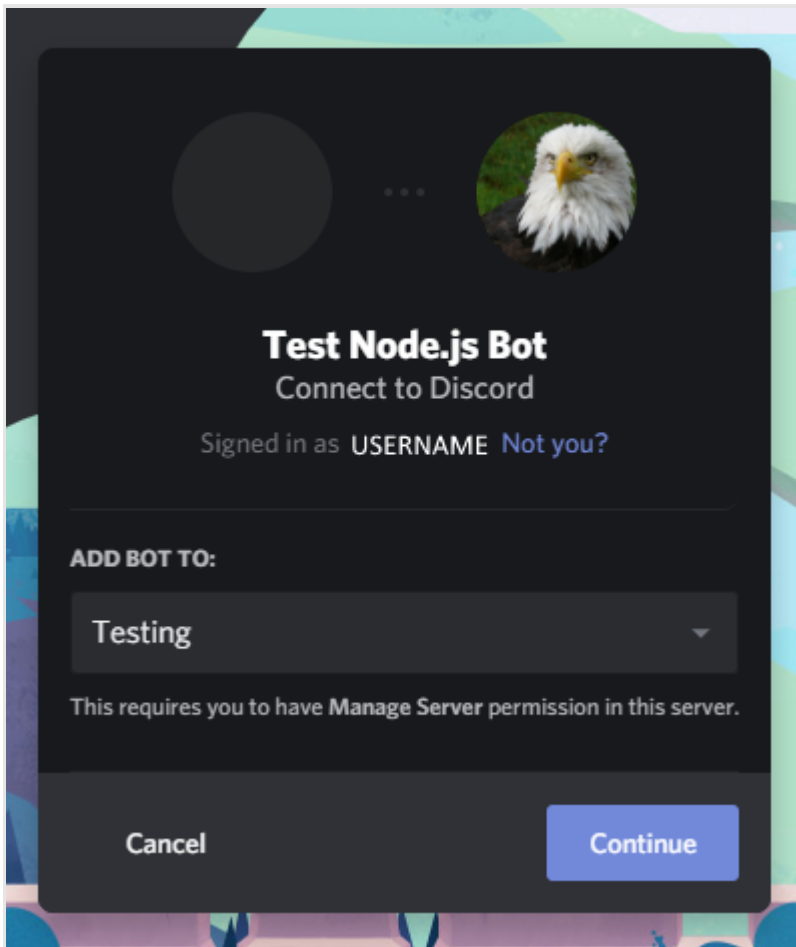
Puede modificar el nombre o la imagen de su bot, aquí, en el panel. También necesita copiar el token de autenticación del bot haciendo clic en **Haga clic para mostrar Token** y copiando el token que aparece.

Advertencia: Nunca comparta ni suba su token de bot, ya que permite a cualquier persona iniciar sesión en su bot.

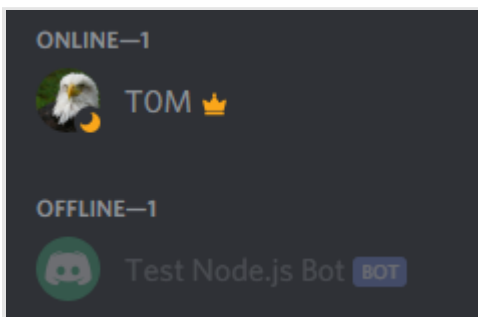
Ahora, necesita crear una invitación que le permita agregar a los servidores de Discord el bot probar el bot. Primero, vaya a la pestaña **OAuth2** del panel de aplicación. Para crear una invitación, desplácese hacia abajo y seleccione **bot** en **alcances**. También debe establecer permisos para controlar las acciones que puede realizar su bot en los servidores. A los efectos de este tutorial, seleccione **Administrador**, que le dará a su bot permiso para realizar casi todas las acciones en los gremios. Copie el enlace con el botón **Copiar**.



A continuación, añada el bot a un servidor. Siga el enlace de invitación que acaba de crear. Puede agregar el bot a cualquier servidor que posea, o en el que tenga permisos de administrador, desde el menú desplegable.



Ahora haga clic en **Continuar**. Asegúrese de tener marcada la casilla junto a **Administrador**: esto concederá los permisos de administrador de bot. A continuación, haga clic en **Autorizar**. Discord le solicitará que resuelva un CAPTCHA antes de que el bot se una al servidor. Ahora, tendrá el bot de Discord en la lista de miembros en el servidor al que añadió el bot debajo de **sin conexión**.



Ha creado correctamente un bot de Discord y lo ha añadido a un servidor. A continuación, escribirá un programa para iniciar sesión en el bot.

Paso 2: Crear su proyecto

En este paso, configurará el entorno de codificación básico donde creará su bot e iniciará sesión en el bot de forma programática.

Primero, debe configurar una carpeta de proyecto y los archivos de proyecto necesarios para el bot.

Cree su carpeta de proyecto:

```
mkdir discord-bot
```

•

[Copy](#)

Vaya a la carpeta de proyecto que acaba de crear:

```
cd discord-bot
```

•

[Copy](#)

A continuación, utilice su editor de texto para crear un archivo llamado `config.json` para almacenar el token de autenticación de su bot:

```
nano config.json
```

•

[Copy](#)

A continuación, añada el siguiente código al archivo config, sustituyendo el texto resaltado por el token de autenticación de su bot:

```
config.json
```

```
{  
  "BOT_TOKEN": "YOUR BOT TOKEN"  
}
```

[Copy](#)

Guarde el archivo y ciérrelo.

A continuación, creará un archivo `package.json` que almacenará los detalles de su proyecto e información sobre las dependencias que utilizará para el proyecto. Creará un archivo `package.json` ejecutando el siguiente comando `npm`:

```
npm init
```

•

[Copy](#)

npm le solicitará varios datos sobre su proyecto. Si desea obtener información sobre cómo completar las preguntas, puede leer sobre ellos en [Cómo usar módulos Node.js con npm y package.json](#).

Ahora, instalará el paquete `discord.js` que utilizará para interactuar con la API de Discord. Puede instalar `discord.js` a través de npm con el siguiente comando:

```
npm install discord.js
```

•

[Copy](#)

Ahora ha configurado el archivo de configuración y ha instalado la dependencia necesaria, está listo para comenzar a crear su bot. En una aplicación en el mundo real, un bot grande se dividiría en muchos archivos, pero a efectos de este tutorial, el código para su bot estará en un solo archivo.

Primero, cree un archivo llamado `index.js` en la carpeta `discord-bot` para el código:

```
nano index.js
```

•

[Copy](#)

Comience a codificar el bot pidiendo la dependencia de `discord.js` y el archivo config con el token de bot:

```
index.js
const { Client, Intents } = require('discord.js');
const client = new Client({ intents: [Intents.FLAGS.GUILDS, Intents.FLAGS.GUILD_MESSAGES] });
const config = require("./config.json");
client.login(config.BOT_TOKEN)
```

[Copy](#)

A continuación, añada las siguientes dos líneas de código:

```
index.js
...
const { Client, Intents } = require('discord.js');
const client = new Client({ intents: [Intents.FLAGS.GUILDS, Intents.FLAGS.GUILD_MESSAGES] });
```



```
const config = require("./config.json");
client.login(config.BOT_TOKEN)
```

Guarde y cierre su archivo.

La primera línea de código crea un nuevo `Discord.Client` y lo asigna a la constante `client`. Este cliente es en parte cómo interactúa con la API de Discord y cómo le notificará eventos como mensajes nuevos. El cliente, en efecto, representa el bot de Discord.

La segunda línea de código utiliza el método `login` en `client` para iniciar sesión en el bot de Discord que creó, usando el token en el archivo `config.json` como contraseña. El token permite a la API de Discord saber para qué bot es el programa y que está autenticado para usar el bot.

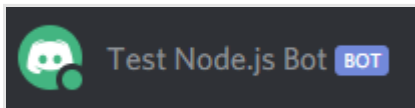
Ahora, ejecute el archivo `index.js` usando Node:

```
node index.js
```

•

[Copy](#)

El estado de su bot cambiará a “en línea” en el servidor de Discord al que lo añadió.



Ha configurado correctamente un entorno de codificación y ha creado el código básico para iniciar sesión en un bot de Discord. En el siguiente paso, manejará los comandos de usuario y hará que su bot realice acciones, como enviar mensajes.

Paso 3: Cómo manejar su primer comando de usuario

En este paso, creará un bot que puede manejar los comandos de usuario. Implementará su primer comando de `ping`, que responderá con `"pong"` y el tiempo que toma para responder al comando.

Primero, necesita detectar y recibir cualquier mensaje que los usuarios envíen, para que pueda procesar cualquier comando. Con el método `on` en el cliente de Discord, Discord le enviará una notificación sobre eventos nuevos. El método `on` toma dos argumentos: el nombre de un evento a esperar y una función a ejecutar cada vez que se produce ese evento. Con este método, puede esperar el evento `message`: se producirá cada vez que se envía un mensaje a un gremio donde el bot tiene permiso para ver mensajes. Por tanto, vamos a crear una función, que se ejecuta cada vez que se envía un mensaje, para procesar comandos.

Primero, abra su archivo:

`nano index.js`

•

[Copy](#)

Añada el siguiente código a su archivo:

```
index.js
...
const { Client, Intents } = require('discord.js');
const client = new Client({ intents: [Intents.FLAGS.GUILDS, Intents.FLAGS.GUILD_MESSAGES] });
const config = require("./config.json")
```

[Copy](#)

Esta función, que se ejecuta en el evento `message`, toma `message` como un parámetro. `message` tendrá el valor de una instancia de mensaje de Discord.js, que contiene información sobre el mensaje enviado y los métodos para ayudar a responder al bot.

Ahora, añada la siguiente línea de código a su función de manejo de comandos:

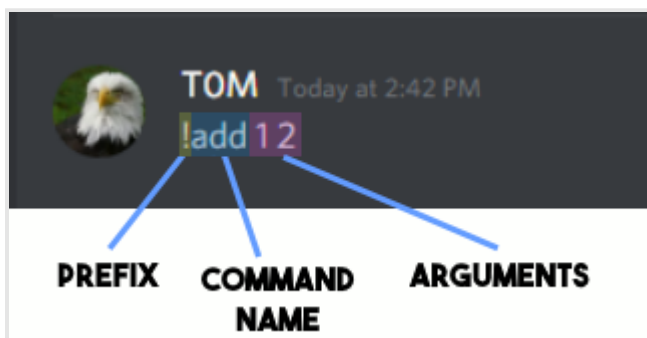
```
index.js
...
client.on("message", function(message) {
  if (message.author.bot) return;
});
...
```

[Copy](#)

Esta línea comprueba si el autor del mensaje es un bot y, si es así, deja de procesar el comando. Esto es importante, ya que generalmente no quiere procesar o responder a los mensajes de bots. Por lo

general, los bots no necesitan o no quieren usar nuestro bot, por lo que ignorar sus mensajes ahorra potencia de procesamiento y ayuda a evitar respuestas accidentales.

Ahora, escribiré un controlador de comandos. Para ello, es bueno entender el formato habitual de un comando de Discord. Normalmente, la estructura de un comando de Discord contiene tres partes en el siguiente orden: un prefijo, un nombre de comando y (a veces) argumentos de comandos.



- **Prefijo:** el prefijo puede ser cualquier cosa, pero, por lo general, es un signo de puntuación o una frase abstracta que normalmente no estaría al principio de un mensaje. Esto significa que, cuando incluya el prefijo al inicio del mensaje, el bot sabrá que la intención de este comando es para que un bot lo procese.
- **Nombre de comando:** el nombre del comando que el usuario quiere usar. Esto significa que el bot puede soportar múltiples comandos con diferentes funciones y permitir a los usuarios elegir entre ellos al proporcionar un nombre de comando diferente.
- **Argumentos:** a veces, si el comando requiere o utiliza información adicional del usuario, este puede proporcionar argumentos después del nombre de comando, con cada argumento separado por un espacio.

Nota: No hay estructura de comandos forzada, y los bots pueden procesar los comandos como quieran, pero la estructura que se presenta aquí es una estructura eficiente que la gran mayoría de bots utilizan.

Para comenzar a crear un analizador sintáctico de comandos que maneje este formato, añade las siguientes líneas de código a la función de manejo de mensajes:

index.js

```
...  
const prefix = "!";
```

```
client.on("message", function(message) {  
  if (message.author.bot) return;  
  if (!message.content.startsWith(prefix)) return;  
});  
...
```

[Copy](#)

Añada la primera línea de código para asignar el valor "!" a la constante `prefix`, que utilizará como prefijo del bot.

La segunda línea de código que añade comprueba si el contenido del mensaje que el bot está procesando comienza con el prefijo que configuró y, si no lo hace, deja de procesarlo.

Ahora, debe convertir el resto del mensaje en un nombre de comando y cualquier argumento que pueda existir en el mensaje. Añada las siguientes líneas resaltadas:

```
index.js  
...  
client.on("message", function(message) {  
  if (message.author.bot) return;  
  if (!message.content.startsWith(prefix)) return;  
  
  const commandBody = message.content.slice(prefix.length);  
  const args = commandBody.split(' ');  
  const command = args.shift().toLowerCase();  
});  
...
```

[Copy](#)

Utilice la primera línea aquí para eliminar el prefijo del contenido de mensaje y asignar el resultado a la constante `commandBody`. Esto es necesario, ya que no quiere incluir el prefijo en el nombre de comando analizado.

La segunda línea toma el mensaje con el prefijo eliminado y utiliza el método `split` en él, con un espacio como separador. Esto lo divide en una matriz de subcadenas y hace una división dondequiera que haya un espacio. Esto crea una matriz que contiene el nombre de comando y, por tanto, si se incluye en el mensaje, cualquier argumento. Asigna esta matriz a la constante `args`.

La tercera línea elimina el primer elemento de la matriz de `args` (que será el nombre de comando que se proporciona), lo convierte en minúscula y, luego, lo asigna a la constante `command`. Esto le permite aislar el nombre de comando y dejar solo argumentos en la matriz. También utiliza el método `toLowerCase`, ya que los comandos no suelen distinguir entre minúsculas y mayúsculas en bots de Discord.

Completó la construcción de un analizador de comandos, implementó un prefijo requerido y obtuvo el nombre de comando y cualquier argumento de los mensajes. Ahora, implementará y creará el código para los comandos específicos.

Añada el siguiente código para comenzar a implementar el comando `ping`:

```
index.js
...
const args = commandBody.split(' ');
const command = args.shift().toLowerCase();

if (command === "ping") {

}
});
...
```

[Copy](#)

Esta instrucción `if` comprueba si el nombre de comando que analizó (asignado a la constante `command`) coincide con `"ping"`. Si lo hace, indica que el usuario quiere usar el comando `"ping"`. Anidará el código para el comando específico dentro del bloque de instrucción `if`. Repetirá este patrón para otros comandos que desee implementar.

Ahora, puede implementar el código para el comando `"ping"`:

```
index.js
...
if (command === "ping") {
  const timeTaken = Date.now() - message.createdTimestamp;
  message.reply(`Pong! This message had a latency of ${timeTaken}ms.`);
}
...
```

[Copy](#)

Guarde y cierre su archivo.

Añada el bloque de comandos `"ping"` que calcula la diferencia entre el tiempo actual, que se encuentra usando el método `now` en el objeto `Date` y la marca de tiempo cuando el mensaje se creó en milisegundos. Esto calcula cuánto tiempo tardó el mensaje en procesarse y el `"ping"` del bot.

La segunda línea responde al comando del usuario usando el método `reply` en la constante `message`. El método `reply` avisa (que notifica al usuario e indica el mensaje para el usuario especificado) al usuario que invocó el comando, seguido por el contenido proporcionado como el primer argumento al método. Proporciona una plantilla literal que contiene un mensaje y el ping calculado como la respuesta que el método `reply` utilizará.

Esto concluye la implementación del comando `"ping"`.

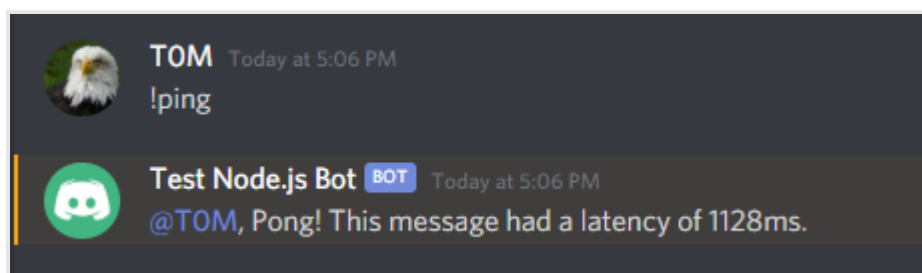
Ejecute su bot usando el siguiente comando (en la misma carpeta que `index.js`):

```
node index.js
```

•

[Copy](#)

Ahora puede usar el comando `"! ping"` en cualquier canal que el bot pueda ver y enviar un mensaje, lo que resulta en una respuesta.



Ha creado correctamente un bot que puede manejar los comandos de usuario y ha implementado su primer comando. En el siguiente paso, continuará desarrollando su bot implementando un comando sum.

Paso 4: Cómo implementar el comando sum

Ahora, extenderá su programa implementando el comando `"! sum"`. El comando tomará cualquier número de argumentos y los agregará juntos, antes de devolver la suma de todos los argumentos al usuario.

Si su bot de Discord aún se está ejecutando, puede detener su proceso con `CTRL + C`.

Abra su archivo `index.js` de nuevo:

```
nano index.js
```

•

[Copy](#)

Para comenzar a implementar el comando `"! sum"`, usará un bloque `else-if`. Después de comprobar el nombre de comando ping, comprobará si el nombre de comando es igual a `"sum"`. Usamos un bloque `else-if`, ya que solo un comando se procesará a la vez, de forma que si el programa coincide con el nombre de comando `"ping"`, no tiene que verificar el comando `"sum"`. Añada las siguientes líneas resaltadas a su archivo:

index.js

```
...
if (command === "ping") {
  const timeTaken = Date.now() - message.createdTimestamp;
  message.reply(`Ping! This message had a latency of ${timeTaken}ms.`);
}

else if (command === "sum") {

}
});
...
```

[Copy](#)

Puede comenzar a implementar el código para el comando `"sum"`. El código para el comando `"sum"` entrará en el bloque `else-if` que acaba de crear. Ahora, añada el siguiente código:

index.js

```
...
else if (command === "sum") {
  const numArgs = args.map(x => parseFloat(x));

```

```
const sum = numArgs.reduce((counter, x) => counter += x);
message.reply(`The sum of all the arguments you provided is ${sum}!`);
}
...
```

[Copy](#)

Utilice el método `map` de la lista de argumentos para crear una nueva lista usando la función `parseFloat` en cada elemento de la matriz de `args`. Esto crea una nueva matriz (asignada a la constante `numArgs`) en la que todos los elementos son números en vez de cadenas. Esto significa que más tarde puede encontrar correctamente la suma de los números sumándolos.

La segunda línea utiliza el método `reduce` en la constante `numArgs` que proporcionan una función que suma todos los elementos de la lista. Asignó la suma de todos los elementos en `numArgs` a la constante `sum`.

A continuación, utilice el método `reply` en el objeto de mensaje para responder al comando del usuario con una plantilla literal, que contiene la suma de todos los argumentos que el usuario envía al bot.

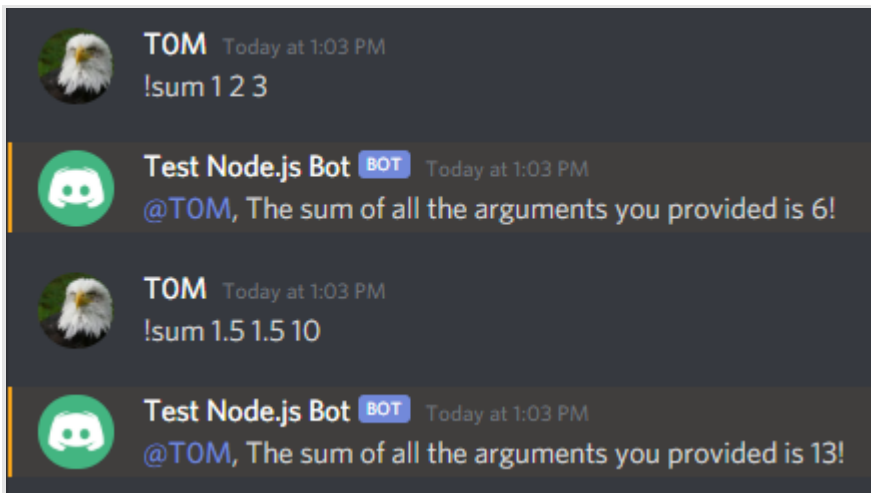
Esto concluye la implementación del comando `"sum"`. Ahora, ejecute su bot usando el siguiente comando (en la misma carpeta que `index.js`):

```
node index.js
```

-

[Copy](#)

Ahora, puede usar el comando `"! sum"` en cualquier canal que pueda ver el bot y enviar mensajes.



A continuación, se muestra una versión completa de la secuencia de comandos del bot `index.js`:

index.js

```
const { Client, Intents } = require('discord.js');
const client = new Client({ intents: [Intents.FLAGS.GUILDS, Intents.FLAGS.GUILD_MESSAGES] });
const config = require("../config.json")

const prefix = "!";

client.on("message", function(message) {
  if (message.author.bot) return;
  if (!message.content.startsWith(prefix)) return;

  const commandBody = message.content.slice(prefix.length);
  const args = commandBody.split(' ');
  const command = args.shift().toLowerCase();

  if (command === "ping") {
    const timeTaken = Date.now() - message.createdTimestamp;
    message.reply(`Pong! This message had a latency of ${timeTaken}ms.`);
  }

  else if (command === "sum") {
    const numArgs = args.map(x => parseFloat(x));
    const sum = numArgs.reduce((counter, x) => counter += x);
    message.reply(`The sum of all the arguments you provided is ${sum}!`);
  }
});

client.login(config.BOT_TOKEN);
```

[Copy](#)

En este paso, ha desarrollado aún más su bot de Discord implementando el comando `sum`.

Conclusión

Ha implementado correctamente un bot de Discord que puede manejar varios comandos de usuario y argumentos de comandos. Si desea ampliar su bot, posiblemente pueda implementar más comandos o probar más partes de la API de Discord para crear un bot de Discord potente. Puede revisar la [documentación de Discord.js](#) o la [documentación de la API de Discord](#) para ampliar sus conocimientos de la API de Discord.

Al crear bots de Discord, siempre debe tener en cuenta los [términos de servicio de la API de Discord](#), que describe cómo los desarrolladores deben usarla. También puede leer [este conjunto de directrices](#) sobre cómo implementar mejor un bot de Discord y proporciona consejos sobre cómo diseñar bots de Discord. Si desea obtener más información sobre Node.js consulte nuestra serie [Cómo crear códigos en Node.js](#).