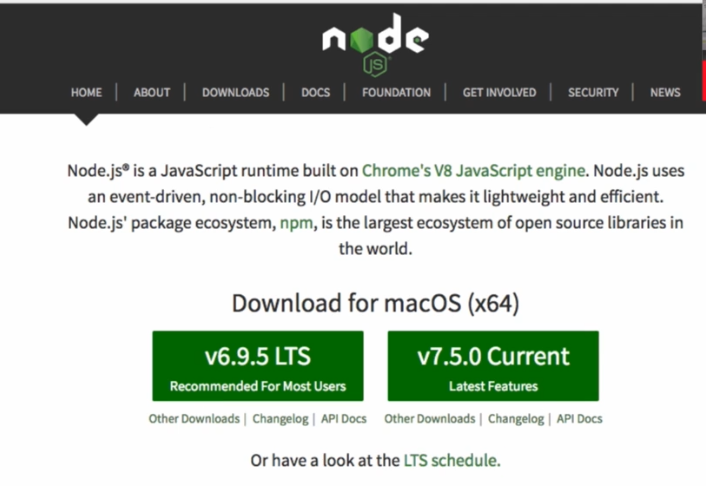
***Prework y Herramientas de un Dev.***

1. **Instalación de Node.**

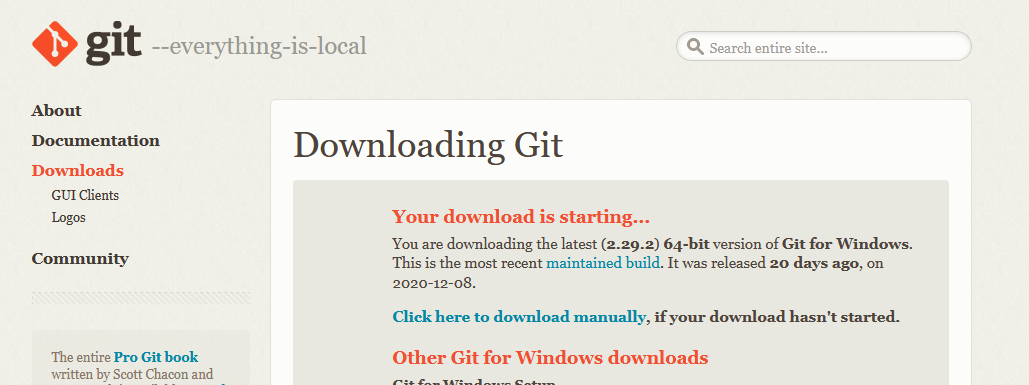
Lo primero que debemos instalar en nuestra maquina para desarrollar es descargar **Node.**



1. **Instalación de Git y Terminal.**

Otra de las herramientas que vamos a tener que usar es Git como repo de nuestro código. En su instalación también descargaremos **Git Bash** como ***terminal*** que vamos a tener que usar eventualmente.

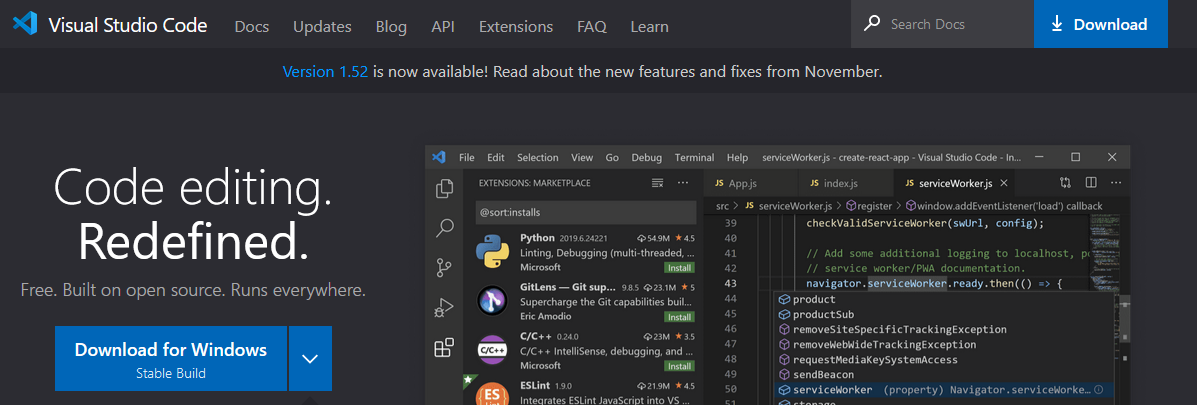
[**https://git-scm.com/download/win**](https://git-scm.com/download/win)



1. **Instalación de un editor de código (VS Code).**

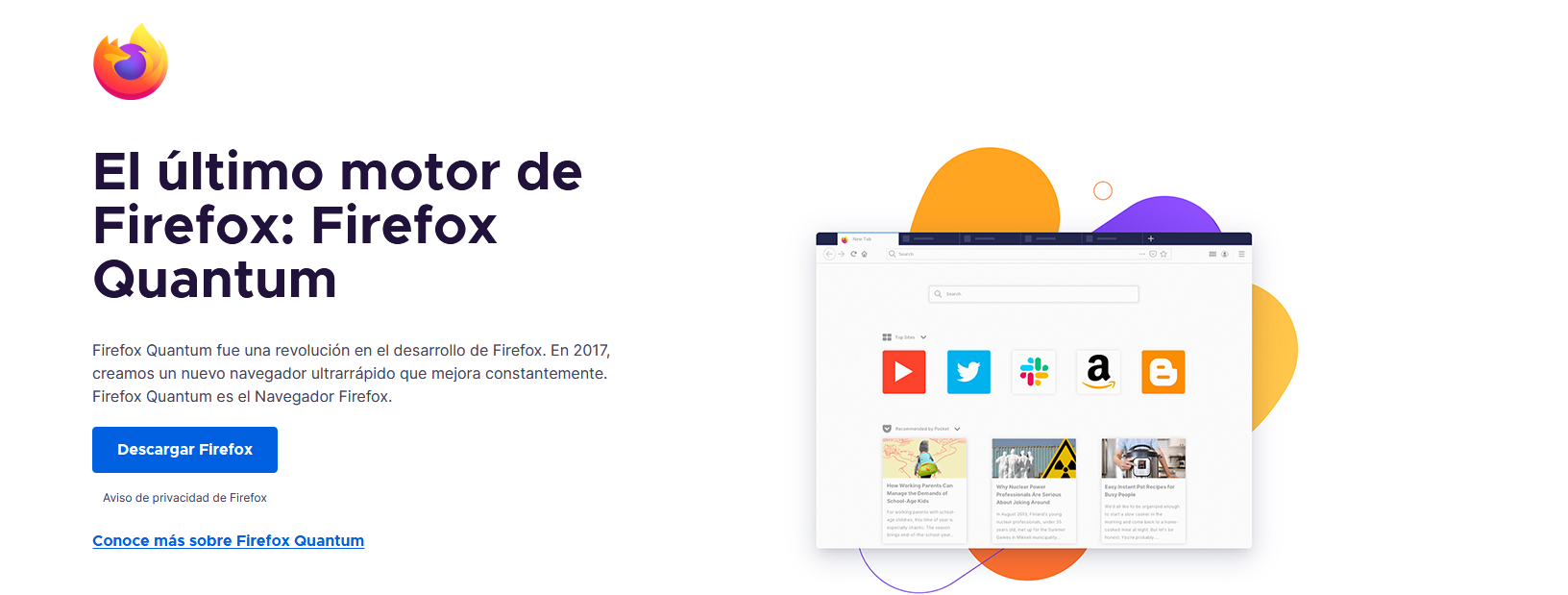
Para editar código tendremos disponibilidad de varios editores, sin embargo en este caso escogeremos VS Code como el editor preferido por sus extensiones y gran experiencia para generar código.

[***https://code.visualstudio.com/***](https://code.visualstudio.com/)



1. **Instalación de un navegador.**

Para desarrollo usaremos el navegador para desarrolladores ***Firefox Quantum.*** Sin embargo podremos usar cualquier navegador como Google Chrome.



**Configurando una aplicación en React JS.**

***Teniendo previamente instalado node en nuestro equipo…***

**Instalación de React.**

Luego de la instalación de Node vamos a poder llamar paquetes con **npm**. En este caso queremos instalar React, su instalación global en el equipo será con:

**npm install -g create-react-app**

**Luego de su instalación, para su uso en desarrollo creamos el proyecto así:**

**create-react-app *nombre\_proyecto***

**Después de que se cree nuestra app, nos cambiamos a la carpeta *cd nombre\_proyecto* y luego iniciamos el servidor con npm run start.**

**Si no queremos instalar React globalmente:**

**Sin embargo hay otra forma de usar react sin tener que instarlo en el equipo. Eliminemos ahora globalmente react:**

**npm uninstall -g create-react-app**

**En las versiones actuales de Node (2020) ya contiene React en las que crearemos nuestros proyectos así:**

**npx create-react-app *nombre\_proyecto***

***cd nombre\_proyecto***

***npm start***

**Introducción a la terminal**

La **terminal** es la herramienta más poderosa para un desarrollador. Cambia un poco dependiendo del sistema operativo, puede ser:

* **Unix**: sistema operativo en el que se basa Mac y Linux.
* **Windows**.



**Crea llaves SSH**

Las llaves SSH nos van a ayudar para autentificarnos con servidores. SSH utiliza criptografía asimétrica, o sea, tenemos dos llaves:

* Pública: la llave pública la podemos compartir por internet.
* Privada: debes tenerla en un sitio seguro y no debe ser compartida.

Tener una llave SSH nos permitirá una conexión fácil y segura con servidores, en el caso de la escuela de JavaScript nos va a servir para conectarnos con GitHub.

Para crear una llave SSH utilizamos el siguiente comando:

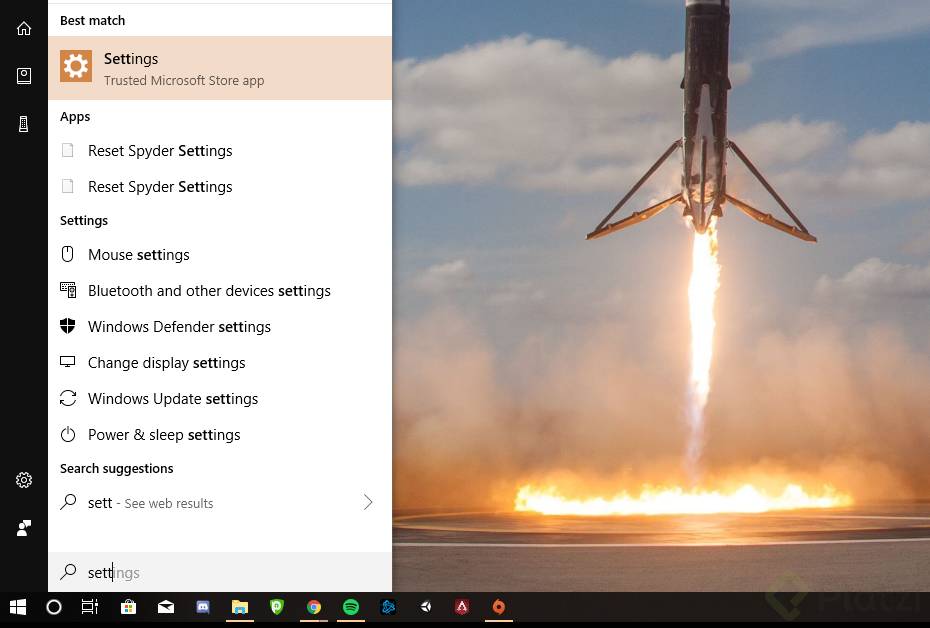
ssh-keygen -t rsa -b 4096 -C llave, puede ser tu correo>

# Instalación de Ubuntu Bash en Windows.

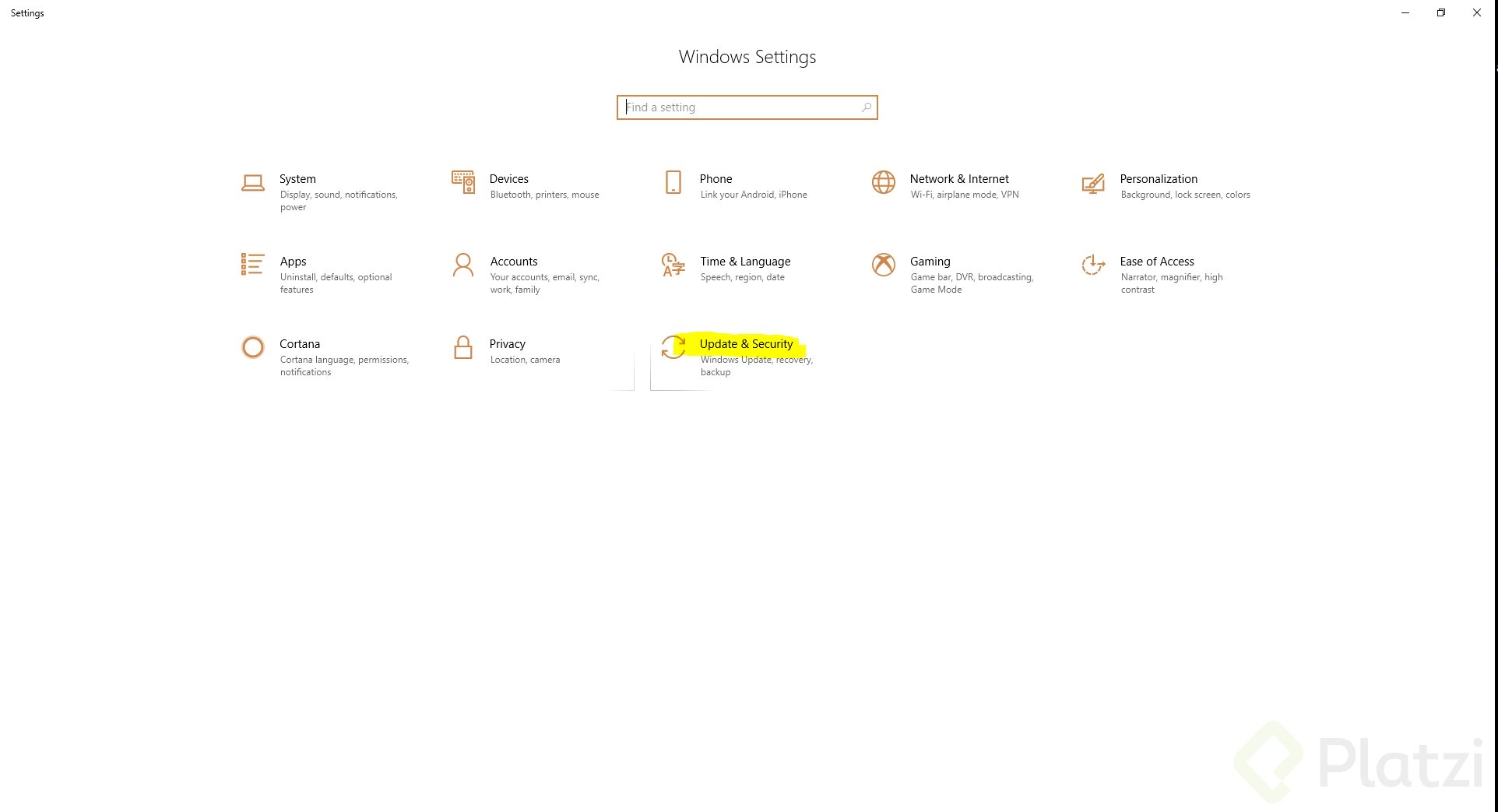
En este tutorial te enseñaré a configurar el Ubuntu dentro de tú Windows 10 para que puedas ejecutar tus códigos de C.

Lo primero que necesitas es que tu computadora tenga instalado Windows 10 de 64 bits y tengas tu sistema operativo actualizado (sobre todo con el “Windows 10 Anniversary Update”

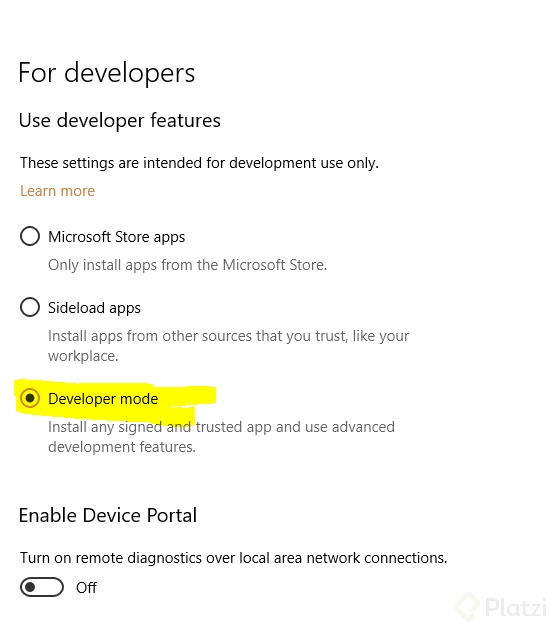
Una vez hayas verificado que tu computadora cumple con los requisitos entra a los settings del sistema (Ajustes)



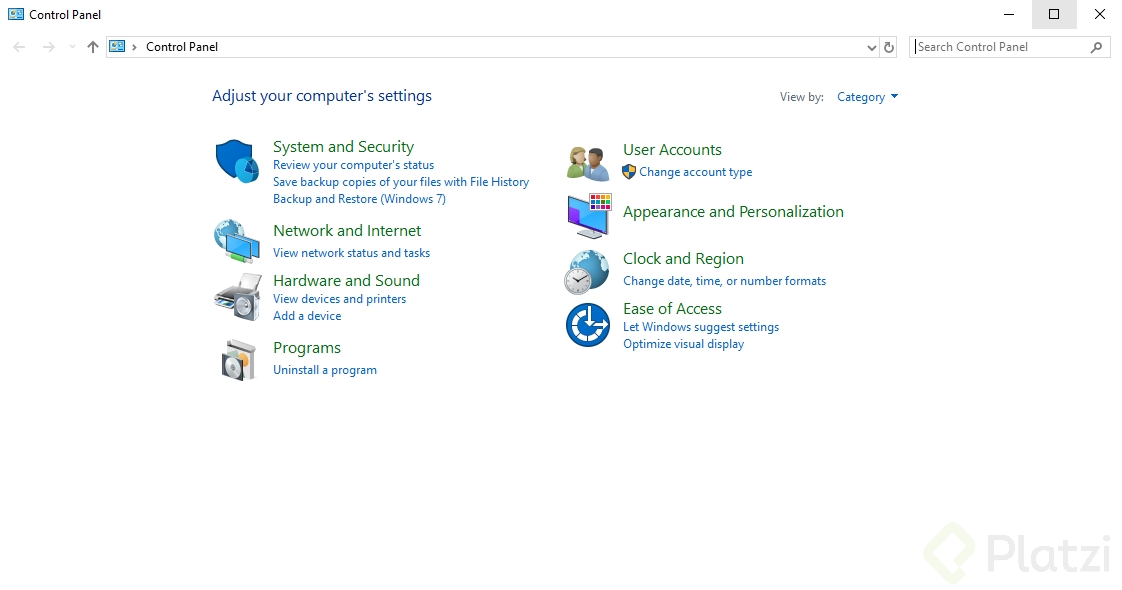
Luego entra a la opción de Actualizaciones y Seguridad



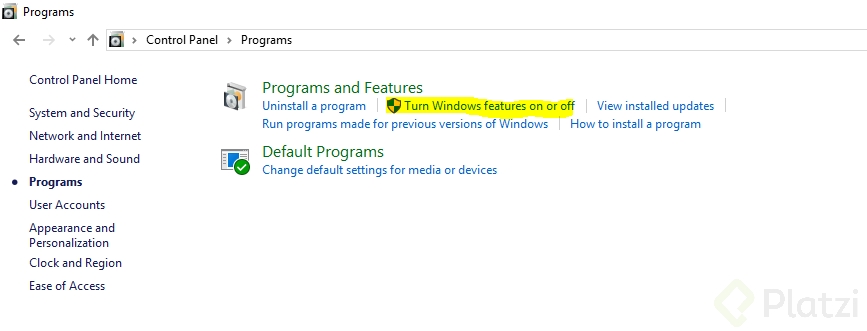
En el menú de la izquierda has click en opciones para desarrolladores y habilita el “Modo Desarrollador”



Después, accede al panel de control y haz click en “Programas”

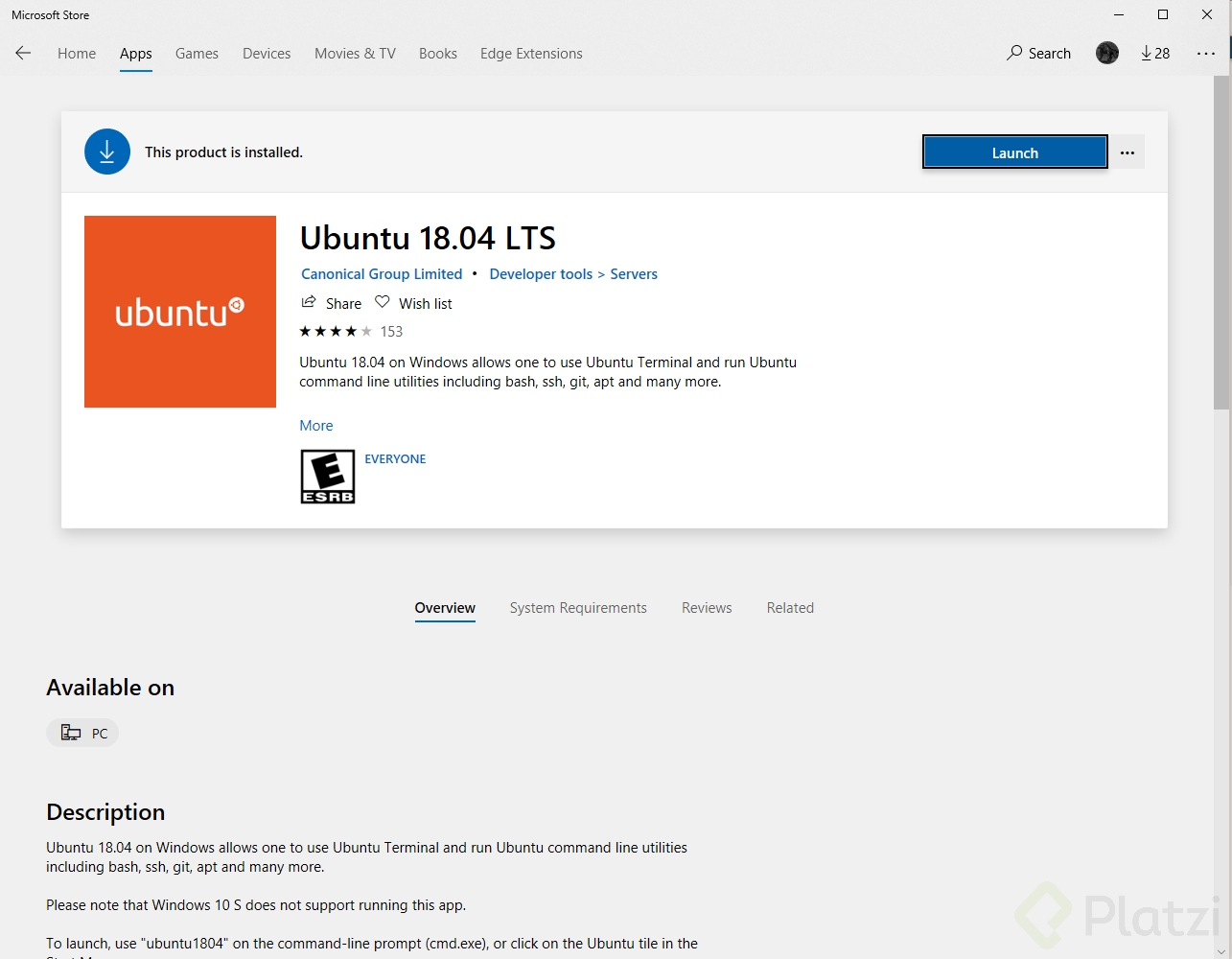


Una vez ahí, haz click en activar o desactivar características de Windows.



Aquí, busca la opción de “Windows Subsystem for Linux” y actívala, instala eso y permite que tu computadora se reinicie.

Luego, entra al menú inicio, escribe bash y sigue los pasos que te indique, en caso de que te diga que no tienes ninguna distribución sólo ve a la tienda de aplicaciones y descargaba Ubuntu para Windows.



En caso de error, ejecutar en Power Shell con permisos de Administrador:

*Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux*

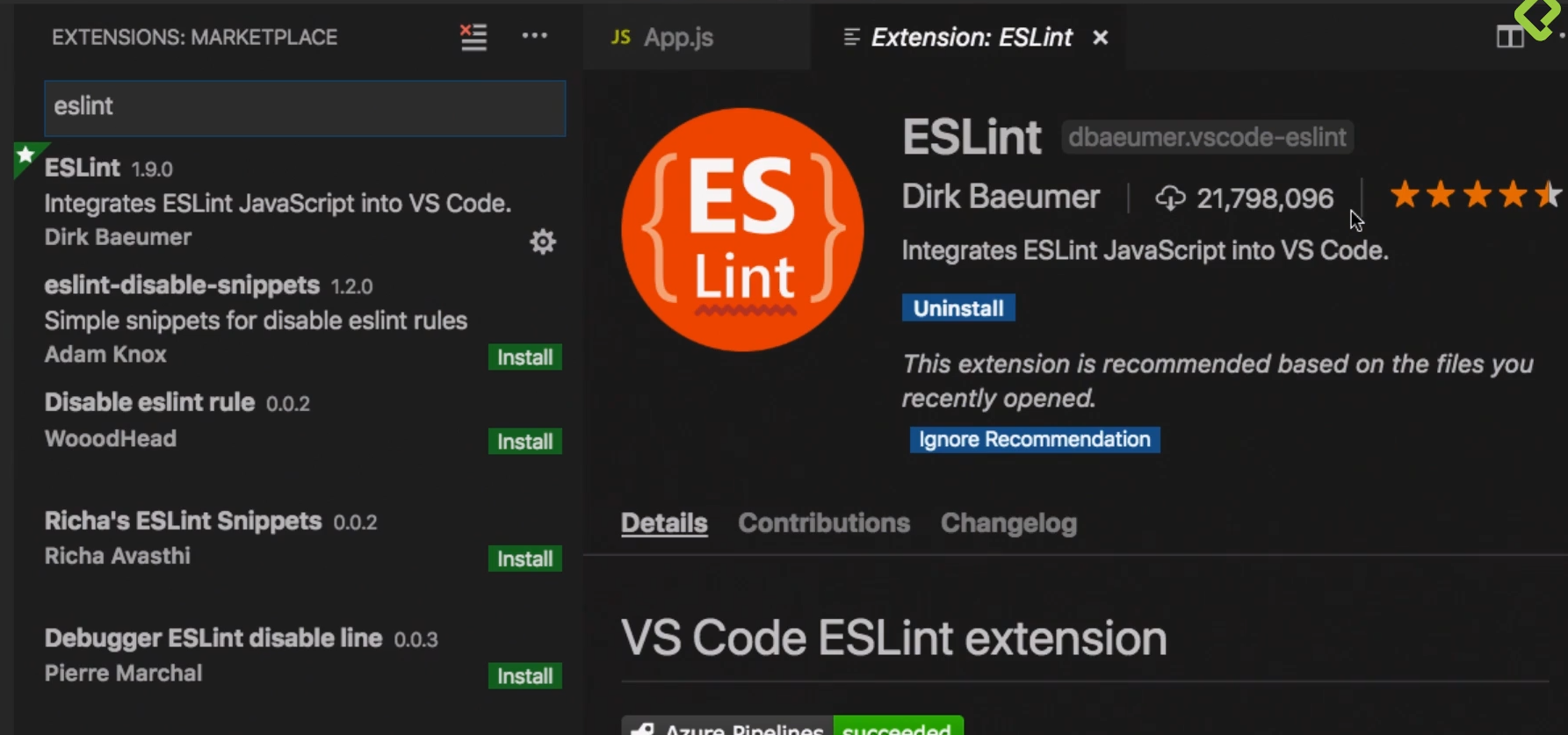
Luego, ejecuta Ubuntu, crea tu usuario y contraseña y estás list@ para continuar.

**VS Code y Extensiones.**

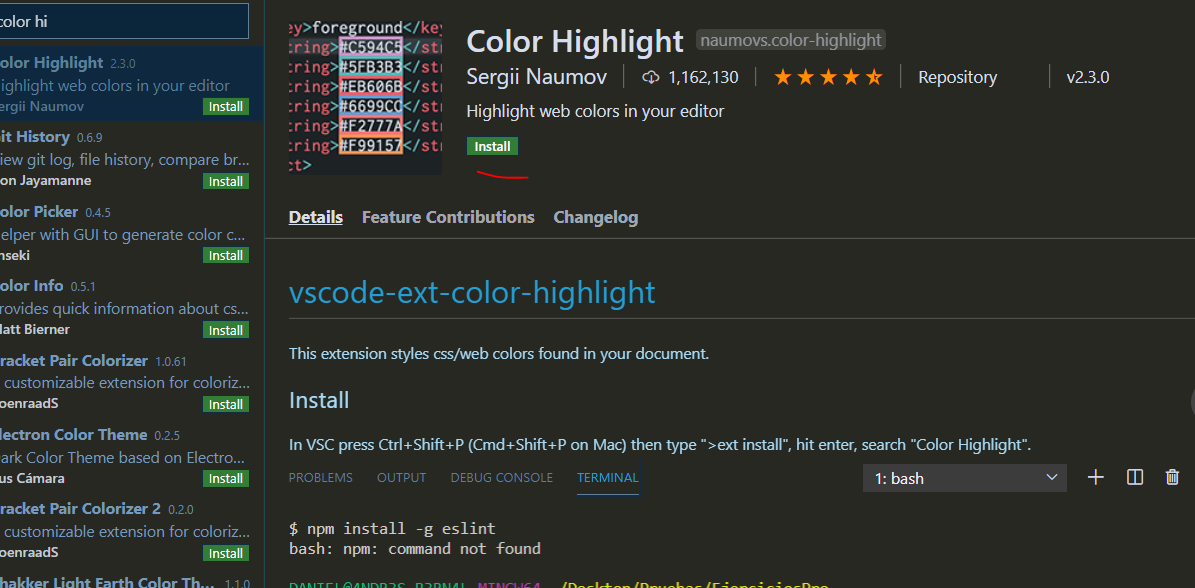
Luego de haber instalado VS Code instalaremos Remote – WSL que nos permitirá usar Windows basado en un ambiente de desarrollo de Linux.



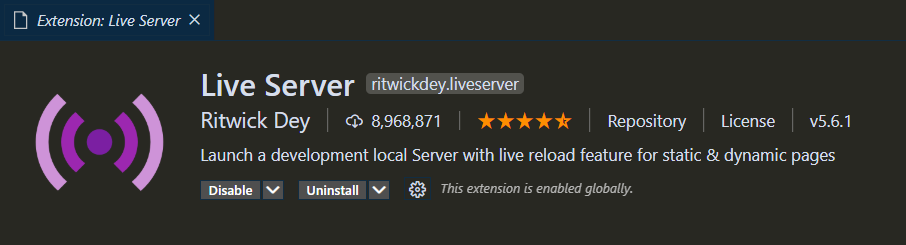
 **ESLint**: es una herramienta de análisis de código estático para identificar patrones problemáticos encontrados en el código JavaScript, o sea, nuestro linter. Debemos instalar y configurar eslint para que siga el estilo de código que le indiquemos.



 **Color Highlight**: resalta el color que estemos escribiendo.



 **LiveServer**: Para poder ver los cambias inmediatos cuando escribimos código podemos usar esta herramienta.



**Configuración de Herramientas Adicionales.**



* ***Hyper.***

Ingresamos al siguiente link y descargamos:

[*https://hyper.is/*](https://hyper.is/)

* ***Oh My ZSH.***

**Comando para instalar zsh:**

1. sudo apt-get install zsh

**Comandos para instalar Oh My Zsh:**

1. curl -Lo install.sh https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh
2. sh install.sh

**Comando para establecer zsh como shell por defecto**

* chsh -s /usr/bin/zsh  
  **En caso de reiniciar la terminal de Hyper y el aspecto no sea el mismo que el profesor intentar lo siguiente desde la terminal de Linux:**
* nano ~/.bashrc
* colocar esta linea **bash -c zsh** al inicio del documento y guardar cambios
* En la misma terminal de Linux ejecutar el comando **chsh -s /usr/bin/zsh** y reiniciar Hyper de nuevo, con estos pasos tu terminal de Hyper ya debería tener el mismo aspecto que la del profesor.

## **Revisión de Node en nuestro sistema.**

En la mayoría de sistemas basados en Unix ya viene instalado por defecto Node, para asegurarnos de que esté instalado debemos irnos a nuestra terminal de comandos y ejecutar:  
$ node -v

Esto nos debería mostrar la versión de node que tenemos instalados en el sistema, por ejemplo:  
$ node -v v12.4.0

Si la respuesta que obtenemos es:  
$ node -v command not found: node

Debemos instalarlo

## **Instalación de Node en Linux**

Dependiendo de tu distribución de Linux deberás ejecutar comandos distintos, esto porque entre distribuciones cambiar el gestor de paquetes:  
En distribuciones basadas en Debian y Ubuntu debes ejecutar:

$ sudo apt update

$ sudo apt install nodejs

$ sudo apt install npm

En distribuciones basadas en Arch:

$ pacman -S nodejs npm

## **Instalación de Node en Windows:**

Esta es la instalación más sencilla y es una instalación clásica en Windows, únicamente descargamos un programa y le damos continuar, o si prefieres configuras la instalación según las opciones que están disponibles. El programa se descarga desde acá <https://nodejs.org/en/#download> y seleccionas la versión que desees (recomendada la versión igual o superior a las 12)

## **Cómo ejecutar Node**

Una vez se tenga instalado Node en el sistema podemos hacer uso de él, en esta clase haremos un uso básico de sus comandos, a lo largo de la Escuela de JavaScript será utilizado. Lo primero que haremos será ejecutarlo y escribir un Hola mundo. En la terminal haremos lo siguiente:  
$ node  
> console.log('Hola mundo')  
Hola mundo  
>

Al escribir node se nos abrirá un shell interactivo donde podremos escribir código en JavaScript. Esta herramienta es esencial en el desarrollo porque es aquí donde podremos probar funcionalidades antes de insertarlas en nuestro proyecto.

## **Cómo utilizar npm**

npm es el manejador de paquetes de Node con él podemos instalar dependencias a nuestro proyecto o instalar programas globalmente en nuestro sistema. A lo largo de este curso y de toda la Escuela de JavaScript npm será quien nos permita correr los proyectos e instalar nuestras dependencias.

# Instalación de Node.js (Opcional)

# Dentro de Ubuntu ejecutamos el siguiente comando:

# curl -sL <https://deb.nodesource.com/setup_12.x> | sudo -E bash –

# En caso de tener el siguiente error consultar el link:

# 

Consultar:

<https://medium.com/@ab.kotecha/successfully-install-the-latest-nodejs-on-the-latest-ubuntu-on-wsl-v1-fix-gpg-key-18d532250219>

* Ejecutamos el comando

*sudo apt-get install -y nodejs*

* Para validar que todo se encuentre correctamente debe mostrarnos la versión:



# Herramientas de desarrollo Backend: JSON Viewer y Postman.

# <https://www.postman.com/downloads/>

# 

# Cómo crear un buen README.md y sintaxis de MarkDown

El README es el archivo en el cual hacemos la descripción del proyecto, ya sea open source o privados es importante tener un buen README. Este archivo se escribe con formato MarkDown, esto es lo primero que veremos en esta clase.

## Markdown

Es un formato de escritura que permite la generación de contenido fácil y rápido, permite generar una salida (por lo general) en formato HTML sin necesidad de aprender a profundidad HTML. Es ampliamente utilizado por su facilidad de generar texto enriquecido.

### Encabezados:

Lo utilizamos para resaltar una parte importante, títulos, subtítulos, etc. Se utiliza el símbolo # para demarcar el inicio de un encabezado.  
# Encabezado nivel 1  
## Encabezado nivel 2  
### Encabezado nivel 3  
#### Encabezado nivel 4  
##### Encabezado nivel 5  
###### Encabezado nivel 6

### Itálicas y negritas

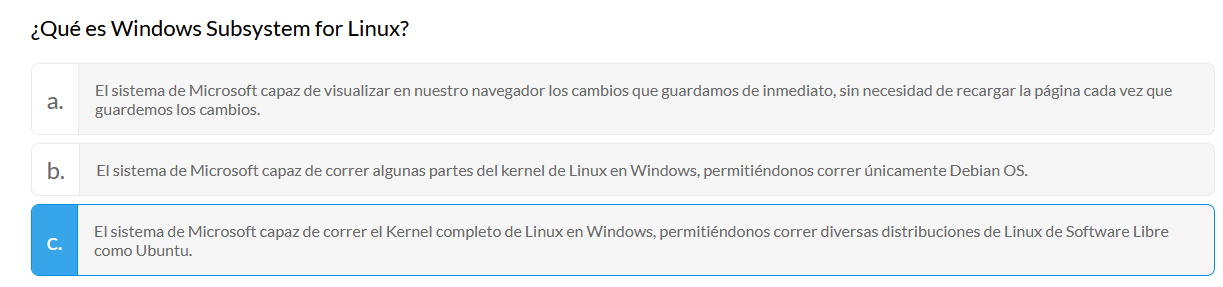
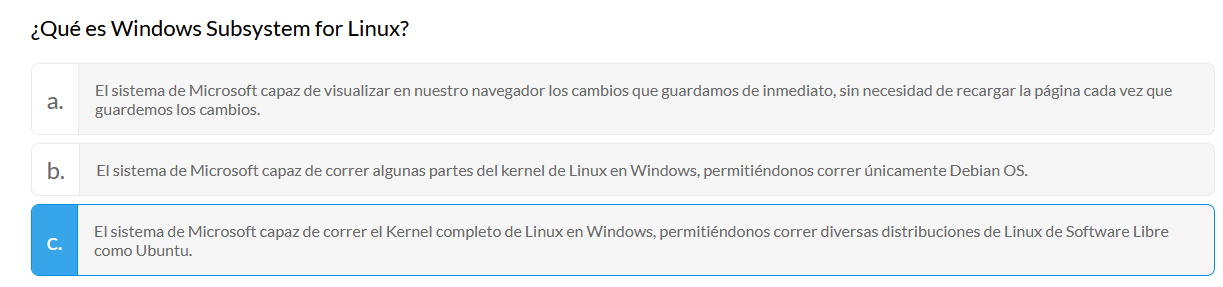
Hay partes en las que necesitaremos hacer énfasis en ciertas palabras, lo común es que utilicemos itálicas y negritas para resaltarlas, en Markdown debemos hacer lo siguiente:  
\*\*Esto es una negrita\*\*  
\*Esto es una itálica\*  
\*\*\_Esto es una negrita con itálica\_\*\*

## Cómo escribir un buen README

No hay un estándar sobre cómo escribir un buen README, cada proyecto es diferente y depende de cada uno. Pero hay ciertas partes que sí o sí debería contener un buen README.

1. Nombre: Especificamos cómo se llama nuestro proyecto.
2. Descripción: es donde diremos para qué exactamente es el proyecto, qué problemas resuelve y cualquier información relevante.
3. Instalación: muestra los pasos específicos para instalar el proyecto. Por lo general se muestra un pedazo del código necesario para la instalación.
4. Cómo usar: describe rápidamente casos de uso en los cuales se puede usar el proyecto, además de mostrar funcionalidades.
5. Cómo contribuir: si es un proyecto open source se describe acá la forma en la que deberían crearse las contribuciones.
6. Licencia: muestra la licencia que tiene el proyecto.  
   En formato markdown podemos escribir cada uno de los ítems de esta manera:





# NPM.

# NPM es un gestor de paquetes, el más popular que tiene JavaScript, donde encontrarás una gran cantidad de recursos para poder implementar en tus proyectos. También vas a poder crear tus propios paquetes y compartirlos con toda la comunidad.

# npm install -g npm@lastest

# Iniciar un proyecto.

# Todo Proyecto debe estar iniciado con git:

# git init

# Luego también tendremos que iniciar npm y generar nuestro archivo *Package.json.* Si usamos este comando con el flag -y vamos a autocompletar los datos generales de este archivo.

# npm init / npm init -y

# 

# Si elegimos escoger este flag -y vamos a poder setear algunos valores como predeterminados como el nombre y el email con npm set init.author.name “*name*”/ set init.autor.email “*email*”

# Instalación de dependencias.

# 

# $ npm install

# Revisa todo el archivo Package.json y actualiza todos los paquetes que encuentre desactualizados.

# Actualizar y eliminar paquetes.

# 

# Extension recomendada:

# 

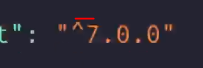
# <https://marketplace.visualstudio.com/items?itemName=eg2.vscode-npm-script>

# Package lock y el uso de símbolos ^ y ~

# 

En el versionado tenemos 3 dígitos que significan:

* **Primer dígito** (***major***): son cambios mayores
* **Segundo dígito (*minor*)**: añaden ciertas funcionalidades pero no representan un gran paso para decir que esta es una versión nueva
* **Tercer dígito (patch)**: estos son patch, bug fixes o cambios menores

\*Cuando tenemos el símbolo (^) caret dentro de la configuración del package.json estamos garantizando que cuando nosotros hagamos una actualización o tengamos un cambio que podamos realizar, vamos a hacer actualización solo de los cambios menores y de los parches o bug fix de este paquete.

\*también podemos establecer una tilde (~) esto quiere decir que vamos a recibir actualización o cambios que son patch o bug fixes.

\*Si queremos tener el control sobre estas actualizaciones garantizando que nos queremos quedar en cierta versión lo mejor es elimina el caret (^)

***Package-lock.json*** a partir de la versión 5 npm encontramos este archivo que nos permite tener ciertas configuraciones la cual nos permite saber que está sucediendo a lo largo de nuestro proyecto sabiendo que versiones, que paquetes y que dependencias se encuentran en este.

# 

# Ejecutar tareas (Scripts).

# 

Los scripts NPM son comandos que podemos establecer para poder ejecutar desde la consola. Estos nos van a dar una serie de salidas según sea el caso.Podemos crear la cantidad de scripts que necesitemos. Estos scripts van a poder correr de forma nativa dentro de nuestra terminal.

**Nota:**

Puedes también especificar scripts con el prefijo “pre” que se ejecutarán automáticamente antes del comando que ejecutaste. Por ejemplo, si defines el comando build y prebuild, cuando corras npm run build el comando prebuild se ejecutará primero. Esto sirve para poder ejecutar tareas que hagan algún tipo de preparación necesaria para correr el comando principal. Sin embargo, hay que hacer notar que si el comando pre falla (retorna un valor que no es 0) el comando principal no se ejecutará. Esto es algo bueno ya que si nuestro proceso de preparación no se realiza de forma exitosa, puede que tengamos problemas al querer ejecutar la tarea principal.

En algunas ocasiones, sin embargo, la tarea previa puede fallar sin que eso afecte la ejecución de la tarea principal. En esos casos puedes usar || exit 0 para retornar 0:

"presass-build": "(rm css/\*.css; rm css/\*.css.map) || exit 0"

Ese es un ejemplo de un comando que hice hace un tiempo. rm puede fallar si el directorio css está vacío, y en ese caso no hay problema, la tarea principal puede funcionar sin ningún problema ya que presass-build tiene el propósito de vaciar ese directorio.

# Solución de problemas.

# 

# <https://snyk.io/>

# Crear un paquete para NPM.

# Primero se crea el archivo index.js en la carpeta src.

# Luego se debe crear una carpeta bin donde se crea el archivo global.js (Configuración que se necesita)

# Y se modifica el package.json colocando la configuración de bin necesaria.

# Probar paquete de NPM localmente.

# 

# Luego de que tengamos ya todo configurado previamente en el package.json ejecutaremos npm link (estando en la carpeta del bin y el package) esto lo que va a hacer es instalar el módulo en nuestro equipo ya que linkea la fuente del código al nombre de ejecución que guardamos en el bin. De esta forma ya tendríamos funcionando nuestro paquete de forma global en nuestro sistema.

# 

# *npm link (sobre la ruta del proyecto)*

# *Otra forma de hacer npm link es: npm install -g /ruta/proyecto*

# 

# Publicar un paquete en NPM.

# 

# Luego de haberlo publicado en NPM crearemos un repositorio en GitHub y lo vincularemos con nuestro proyecto.

# 

# Luego ejecutaremos nuevamente *npm init* y este reconocerá que ya hay un package.json existente pero ahora va a modificar parte de este ya que ahora tenemos nuestro proyecto configurado en GitHub.

# Después de esto actualizaremos su versión en NPM, para ello usaremos un comando que tendrá disponible el (major patch minor “2.0.0”) dependiendo de que ajuste fue el que realizamos.

# *npm version (major/patch/minor)*

# Luego publicamos nuevamente con esa version npm publish

# Link de apoyo:

# <https://medium.com/canariasjs/creando-componentes-en-react-y-publicando-en-npm-16eee85f9fba>

# Importants Questions and Answers.

# 

