GitHub Universe Cloud Skills Challenge

Cuaderno de notas sobre el desafío

Elaborado por:

Marco Silva Huerta

01 de Noviembre de 2023

Índice

1. Introducción a Git

1.1. Importancia del Control de Versiones

El control de versiones es esencial en el desarrollo de software, ya que permite gestionar los cambios en el código fuente y la colaboración entre desarrolladores de manera eficiente y segura. Git es un VCS de código abierto rápido, versátil, muy escalable y gratuito. Su autor principal es Linux Torvalds, creador de Linux.

Imaginemos que estamos escribiendo un reporte de practica con nuestro equipo, seguramente a muchos nos paso que todos nos conectábamos al mismo tiempo escribiendo sobre el documento haciendo un mezcla en todo el texto, agregando referencias o imágenes, incluso hasta borrando cosas de los demás. Usando control de versiones, cada modificación se registra, permitiendo ver quién hizo qué y cuándo.

¿Qué ofrece un control de versiones?

- **Historial de Cambios:** Un controlador de versiones mantiene un registro detallado de todos los cambios realizados en el código a lo largo del tiempo, lo que permite rastrear quién hizo qué y cuándo.
- **Ramificación y Fusiones:** Permite la creación de ramas *(branching)* para trabajar en nuevas características o correcciones sin afectar la versión principal. Posteriormente, se pueden fusionar *(mergear)* estas ramas de manera controlada.
- **Recuperación de Versiones Anteriores:** Si surge un problema en una versión de software, es posible volver a una versión anterior para solucionar el problema rápidamente.
- **Copias de Seguridad:** Los controladores de versiones actúan como una copia de seguridad automática del código fuente. En el caso de Git, cada clon del repositorio es una copia completa del historial, lo que garantiza la seguridad y la disponibilidad de los datos.
- **Revisiones y Pruebas Colaborativas:** Los miembros del equipo pueden realizar revisiones de código de manera conjunta, identificar problemas y realizar pruebas en paralelo, acelerando el proceso de desarrollo.

Control de versiones distribuido

Los sistemas de control de versiones centralizados, como `CVS, SVN` y `Perforce`, almacenan el historial de un proyecto en un solo servidor. Esto puede ser un problema si el servidor falla o está inaccesible. Git es un sistema de control de versiones distribuido, lo que significa que el historial de un proyecto se almacena en el cliente y en el servidor. Esto hace que Git sea más robusto y fiable que los sistemas centralizados.

2. Terminología de Git

Árbol de trabajo

Conjunto de directorios y archivos anidados que contienen el proyecto en el que se trabaja.

Repositorio (repo):

Un repositorio, también conocido como repo, es un directorio situado en el nivel superior de un árbol de trabajo en Git, donde se almacena todo el historial y los metadatos de un proyecto. Puede haber repositorios vacíos, que no forman parte de un árbol de trabajo y se utilizan para compartir o realizar copias de seguridad. Un repositorio vacío generalmente es un directorio con un nombre que termina en `.git`, por ejemplo, project.git.

Inicializar un Repo:

Es el proceso de crear una nueva copia de trabajo de Git y asociarla con un remoto. Este proceso se puede realizar mediante el comando `git init`.

Hash:

Un hash es un número generado por una función hash que representa el contenido de un archivo u otro objeto como un número de dígitos fijo. En Git, se utilizan hashes de 160 bits de longitud. La ventaja de los códigos hash en Git es que permiten verificar si un archivo ha cambiado mediante la comparación de su hash actual con el hash anterior. Si el contenido del archivo no cambia, aunque se modifique la marca de fecha y hora, el hash seguirá siendo el mismo.

Objeto:

Un repositorio de Git contiene cuatro tipos de objetos, cada uno identificado de forma única por un hash SHA-1. Un objeto blob contiene un archivo normal, un objeto árbol representa un directorio y contiene nombres, valores hash y permisos, un objeto de confirmación representa una versión específica del árbol de trabajo, y una etiqueta es un nombre asociado a una confirmación en Git.

Confirmación:

El término *confirmación* puede usarse como un verbo para referirse a la acción de crear un objeto de confirmación en Git. Esto implica guardar los cambios realizados en un proyecto para que otros usuarios puedan acceder a ellos.

Rama:

Una rama en Git es una serie de confirmaciones vinculadas con un nombre. La confirmación más reciente en una rama se conoce como el nivel superior de esa rama. La rama predeterminada, que se crea al inicializar un repositorio, se llama `main` y suele tener el nombre `master` en Git. La rama actual se identifica como `HEAD`. Las ramas son una característica poderosa de Git que permite a los desarrolladores trabajar de forma independiente o colaborativa en diferentes ramas y luego fusionar los cambios en la rama predeterminada.

Remoto:

Un remoto en Git es una referencia con nombre a otro repositorio de Git. Al crear un repositorio, Git generalmente crea un remoto llamado `origin`, que es el remoto predeterminado para las operaciones de envío e incorporación de cambios.

Comandos, subcomandos y opciones:

Las operaciones en Git se realizan mediante comandos, subcomandos y opciones. El comando principal, como git push o git pull, especifica la operación que se desea realizar. Los comandos suelen ir acompañados de opciones que se utilizan con guiones (-) o guiones dobles (–). Por ejemplo, git reset –hard es un comando que implica un subcomando reset con la opción –hard.

Git y GitHub

Git es un sistema de control de versiones distribuido (DVCS) que varios desarrolladores y otros colaboradores pueden usar para trabajar en un proyecto. Proporciona una manera de trabajar con una o varias ramas locales y luego insertarlas en un repositorio remoto.

GitHub es una plataforma en la nube que usa Git como tecnología principal. Simplifica el proceso de colaboración en proyectos y proporciona un sitio web, más herramientas de línea de comandos y un flujo integral que los desarrolladores y usuarios pueden usar para trabajar juntos. GitHub actúa como el repositorio remoto mencionado anteriormente.

2.1. Configuración de Git

- Para comprobar que Git está instalado, usar el comando git –version
- Para configurar Git, se define por variables globales: user.name y user.email. Ambas necesarias para hacer confirmaciones.

```
git config --global user.name <USER_NAME>
git config --global user.email <USER_EMAIL>
```

- Lista detallada de la configuración de GIT git config –list

Configuración del repositorio de Git

1. Cree una carpeta, la cual será el directorio del proyecto y nos movemos a ella

```
mkdir Cats
cd Cats
```

2. Inicializar el repositorio

```
git init -b main
```

3. Usando el comando

```
git status
```

para mostrar el estado del árbol de trabajo

3. Inicio de un proyecto

1. Estando dentro de la carpeta Cats:

   ```
   touch index.html
   ```

   Creación de un archivo
2. Usar `git status` para ver el estado del árbol de trabajo
3. Ahora usar

   ```
   git add .
   ```

   para agregar el nuevo archivo al índice de Git, el punto al final es para agregar todo lo que no tiene seguimiento
4. Volver a usar un `git status`
5. Realización de la primera confirmación
   - Utilizar el comando siguiente para crear otra confirmación:

     ```
     git commit -m "Añadiendo archivo index.html"
     ```

   - `git status` Para ver que todo salió bien
   - `git log` Para mostrar la información del commit
6. Modifique index.html y confirme el cambio.
7. Usando `code index.html` se abre en VSC el archivo, así mismo si se usa `code .` en una carpeta se abre toda la carpeta
8. Modificamos el archivo
9. Ahora usamos el comando:

   ```
   git commit -a -m "Add a heading to index.html"
   ```

   - No se ha usado `git add`
   - Usamos la marca `-a` para agregar los archivos modificados desde la última confirmación. No nuevos
10. Ahora volvemos a modificar el html
11. Usando el comando `git diff` para ver lo que ha cambiado
12. Añadiendo `.gitignore`

13. Con `git add -A` para agregar todos los archivos sin seguimiento
14. Añadiendo subdirectorio
    - Vamos a crear una carpeta

      ```
      mkdir CSS
      git status
      ```

    - Git no considera directorios vacíos
    - El comando `touch` sirve también para actualizar

      ```
      touch CSS/.git-keep
      git add . CSS
      git status
      ```

    - Ahora si la carpeta se ve en el área de trabajo
15. Remplazo de un archivo
    - Eliminar .git-kepp del subdirectorio

      ```
      rm CSS/.git-keep
      cd CSS
      code . site.css
      ```

      Con esto habremos borrado el kepp, cambiado a la carpeta CSS y abierto VSC en el archivo para modificarlo
16. Enumeración de confirmaciones
    - Con el comando `git log` se revisan todas las confirmaciones
    - Con el comando `git log –oneline` se obtiene una lista más simplificada

Corrección de errores simples

Rectificación de una confirmación:

```
git commit --amend --no-edit
```

Permite realizar cambios adicionales en el commit más reciente sin cambiar el mensaje de confirmación. Este comando es útil cuando te das cuenta de que olvidaste incluir algunos cambios en el commit anterior o cuando deseas realizar ajustes sin tener que cambiar el mensaje de confirmación.

Recuperación de un archivo eliminado:

```
git checkout -- <file_name>
```

Se utiliza para descartar los cambios no confirmados en un archivo específico y restaurar ese archivo a su estado tal como se encuentra en el último commit.

Recuperación de archivos: (git reset) También puede eliminar un archivo con git rm. Este comando elimina el archivo en el disco, pero también hace que Git registre su eliminación en el índice.

```
git rm index.html
git checkout -- index.html
```

Para recuperar index.html se usa git reset. Puede usar git reset para anular el almacenamiento provisional de los cambios.

```
git reset HEAD index.html
git checkout -- index.html
```

Reversión de una confirmación: git revert

El comando git revert en Git se utiliza para deshacer un commit anterior, creando un nuevo commit que revierte los cambios realizados en el commit original. A diferencia de git reset, que reescribe la historia y elimina commits, git revert no reescribe la historia, sino que crea un nuevo commit que deshace los cambios del commit anterior.

4. Colaboración con Git

Clonación de un repositorio

En Git, un repositorio se copia al clonarlo mediante el comando git clone. Puede clonar un repositorio independientemente de dónde esté almacenado, siempre que tenga una dirección URL o una ruta de acceso a la que apuntar. En Unix y Linux, la operación de clonación usa vínculos físicos, así que es rápida y usa un espacio mínimo, ya que solo hay que copiar las entradas de directorio, no los archivos.

Repositorio remoto

Cuando se clona repositorio en Git, establece una referencia al repositorio original llamado *origin*. Esto facilita la incorporación y envío de cambios. El comando utilizado es git pull, copia las confirmaciones y objetos nuevos del repositorio remoto al local. A diferencia de otros métodos como scp o Rsync, Git solo examina las confirmaciones, no todos los archivos.

Git guarda la lista de confirmaciones obtenidas, y al usar git pull, solicita al repositorio remoto enviar solo los cambios, incluyendo nuevas confirmaciones y objetos. Estos se agrupan en un archivo llamado paquete y se envían en un lote. Luego, Git actualiza el árbol de trabajo al desempaquetar y combinar estos objetos con las confirmaciones locales.

Es importante destacar que Git solo incorpora o envía cambios cuando el usuario lo indica, a diferencia de sistemas como Dropbox que dependen del sistema operativo para notificar cambios en la carpeta y consultan al servidor sobre posibles modificaciones de otros usuarios.

Creación de solicitudes de incorporación de cambios

git request−pull <inicio> <fin> <repositorio remoto>

Este comando en Git se utiliza para generar y enviar solicitudes formales de incorporación de cambios. Al especificar el rango de cambios entre dos puntos (inicio y fin) y el repositorio remoto destinatario,

se crea una solicitud que incluye detalles sobre las modificaciones propuestas. Esto facilita el proceso de revisión y aceptación por parte del propietario del repositorio remoto, permitiendo una integración eficiente de contribuciones externas al proyecto.

Supongamos que he estado trabajando en una nueva función en mi rama local llamada *nueva-funcion* y quiero solicitar la incorporación de estos cambios al repositorio remoto llamado *repositorio-remoto*. Utilizaría el comando git request-pull de la siguiente manera:

git request-pull

origin/nueva-funcion

https://github.com/usuario/repositorio-remoto.git

origin/nueva-funcion: es la rama que contiene los cambios que deseo incorporar.

https://github.com/usuario/repositorio-remoto.git: es la URL del repositorio.

Este comando generaría un resumen de los cambios entre el estado actual de mi rama *nueva-funcion* y su punto de inicio. Luego, puedo enviar este resumen al propietario del repositorio remoto para que revisen y consideren la incorporación de mis cambios al proyecto principal.

5. Creación y combinación de ramas en Git

Introducción

La gestión eficiente de código es esencial en el desarrollo de software, y Git se ha convertido en una herramienta fundamental en este proceso. La capacidad de trabajar en paralelo en diferentes características o soluciones, sin afectar el código principal, es crucial para mantener la estabilidad y facilitar la colaboración entre desarrolladores. En este contexto, la edición de código mediante la creación de ramas y su posterior combinación en Git ofrece un enfoque estructurado y seguro para implementar nuevas funcionalidades, corregir errores o realizar mejoras en un proyecto.

Ramas en Git

Las ramas en Git son como líneas independientes de desarrollo que nos permiten trabajar en diferentes aspectos de un proyecto de software de manera simultánea. Imagina el código de tu proyecto como un árbol principal; una rama sería una bifurcación que se origina desde el tronco principal. Esto nos brinda la flexibilidad de implementar nuevas características, corregir errores o realizar mejoras sin afectar directamente el código base.

En términos sencillos, las ramas son como versiones paralelas de tu proyecto, donde puedes experimentar y hacer cambios sin preocuparte por afectar el código principal. Esta funcionalidad es fundamental para el trabajo en equipo, ya que diferentes desarrolladores pueden trabajar en ramas separadas, abordando distintas tareas al mismo tiempo sin interferir entre sí.

La utilidad principal de las ramas radica en su capacidad para facilitar el desarrollo colaborativo. Al crear ramas, cada miembro del equipo puede concentrarse en una tarea específica sin temor a afectar el trabajo de los demás. Posteriormente, estas ramas pueden fusionarse de manera ordenada, combinando los cambios realizados de manera coherente en el proyecto principal.

Estructura y nomenclatura de las ramas

Cuando hablamos de la estructura y nomenclatura de las ramas en Git, es importante comprender algunos términos clave que nos ayudarán a orientarnos en nuestro flujo de trabajo. En diferentes plataformas y proyectos, la rama principal suele denominarse main, master o trunk. Esta rama es como el tronco principal de nuestro árbol de desarrollo y suele contener la versión estable y funcional de nuestro proyecto.

Las ramas suelen originarse a partir de la rama principal. Al crear una nueva rama, estamos esencialmente bifurcando el desarrollo para trabajar en una funcionalidad específica o resolver un problema particular. Estas nuevas ramas, también conocidas como ramas secundarias, nos permiten realizar cambios sin afectar directamente el código en la rama principal.

Creación y modificación de ramas

Para crear una rama en Git, es tan sencillo como utilizar el comando git branch seguido del nombre que le quieras dar a tu nueva rama. Por ejemplo, si quiero trabajar en una nueva característica llamada nueva-funcionalidad, escribiría git branch nueva-funcionalidad. Luego, para empezar a trabajar en esa rama, simplemente uso git checkout nueva-funcionalidad o el comando más reciente git switch nueva-funcionalidad. Esto me coloca en mi nueva rama, listo para hacer cambios sin afectar la rama principal.

Cambiar entre ramas es igual de fácil. Si quiero volver a la rama principal, uso el comando git checkout master o git switch master. Es un proceso rápido y me permite alternar entre diferentes tareas o características.

Cuando llega el momento de combinar las ramas, utilizo el comando git merge. Supongamos que estoy en mi rama nueva-funcionalidad y quiero agregar los cambios a la rama principal. Después de hacer mis cambios, vuelvo a la rama principal con git checkout master o git switch master y escribo git merge nueva-funcionalidad. Esto fusiona los cambios de mi rama nueva-funcionalidad con la rama principal.

Ahora, evitar conflictos entre las ramas es clave. Git hace un gran trabajo fusionando automáticamente los cambios cuando no hay conflictos. Sin embargo, si dos ramas modifican la misma parte del código, podría haber un conflicto. Para prevenir esto, es útil realizar fusiones frecuentes desde la rama principal a tu rama de desarrollo para mantenerlas actualizadas y minimizar los posibles conflictos.

Resolución de conflictos de combinación

git merge –abort git reset –hard Los desarrolladores parecen preferir la última opción. Cuando Git detecta un conflicto en las versiones del contenido, inserta ambas versiones del contenido en el archivo. Git usa un formato especial para ayudarle a identificar y resolver el conflicto: corchetes angulares de apertura «“‘¡, guiones dobles (signos igual) ======= y corchetes angulares de cierre ’”»¿. El contenido situado encima de la línea de guiones ======= muestra los cambios en la rama. El contenido que se encuentra debajo de la línea de separación muestra la versión del contenido de la rama en la que intenta realizar la combinación.

6. Introducción a GitHub

Introducción

GitHub proporciona una plataforma para desarrolladores con tecnología de inteligencia artificial para compilar, escalar y entregar software seguro. Tanto si planea nuevas características, corregir errores o colaborar en los cambios, GitHub es donde más de 100 millones de desarrolladores de todo el mundo se unen para crear cosas y mejorarlas aún más.

En este módulo, aprenderá los conceptos básicos de GitHub y comprenderá mejor sus características fundamentales con un ejercicio práctico en un repositorio de GitHub.

¿Qué es GitHub?

GitHub es una plataforma basada en la nube que usa Git, un sistema de control de versiones distribuido, en su núcleo. La plataforma GitHub simplifica el proceso de colaborar en proyectos y proporciona un sitio web, herramientas de línea de comandos y un flujo global que permite a los desarrolladores y usuarios trabajar juntos.

Como hemos aprendido anteriormente, GitHub proporciona una plataforma para desarrolladores con tecnología de inteligencia para crear, escalar y ofrecer software seguro. Vamos a desglosar cada uno de los pilares básicos de la plataforma GitHub Enterprise, inteligencia artificial, colaboración, productividad, seguridad y escala.

INTELIGENCIA ARTIFICIAL

La inteligencia artificial generativa está transformando drásticamente el desarrollo de software a medida que hablamos.

La plataforma GitHub Enterprise mejora la colaboración a través de solicitudes de incorporación de cambios y problemas con tecnología de inteligencia artificial, la productividad a través de Copiloto y la seguridad mediante la automatización de las comprobaciones de seguridad más rápido.

Colaboración

En esencia, colaboración de todo lo que hace GitHub. Sabemos que la colaboración ineficaz da como resultado tiempo y dinero desperdiciados. Lo contrarrestamos con un conjunto de herramientas sin fisuras que permiten colaborar sin esfuerzo.

Los repositorios, las incidencias, las solicitudes de incorporación de cambios y otras herramientas ayudan a los desarrolladores, administradores de proyectos, líderes de operaciones y otros usuarios de la misma empresa a trabajar más rápido, reducir los tiempos de aprobación y enviar más rápidamente.

Productividad

La productividad se acelera con la automatización que proporciona la plataforma GitHub Enterprise. Con las herramientas de CI/CD integradas directamente en el flujo de trabajo, la plataforma ofrece a los usuarios la capacidad de establecer tareas y olvidarlas, cuidar de la administración rutinaria y acelerar el trabajo diario. Esto proporciona a los desarrolladores más tiempo para centrarse en lo que más importa: crear soluciones innovadoras.

Seguridad

GitHub se centra en integrar la seguridad directamente en el proceso de desarrollo desde el principio. La plataforma GitHub Enterprise incluye características de seguridad nativas y de primera entidad que minimizan el riesgo de seguridad con una solución de seguridad integrada. Además, el código permanece privado dentro de su organización y, al mismo tiempo, puede aprovechar las ventajas de la información general de seguridad y Dependabot.

GitHub ha seguido realizando inversiones para asegurarse de que nuestras características estén listas para la empresa. Estamos respaldados por Microsoft, que confía en sectores altamente regulados y cumplen los requisitos de cumplimiento globalmente.

Escala

GitHub es la comunidad de desarrolladores más grande de su tipo. Con datos en tiempo real en más de 100 000 desarrolladores, más de 330 000 repositorios e innumerables implementaciones, hemos podido comprender las necesidades cambiantes de los desarrolladores y realizar cambios en nuestro producto para adaptarnos a ellas.

Esto se ha traducido en una escala increíble que no tiene parangón ni comparación con ninguna otra empresa del planeta. Cada día obtenemos más información de esta impresionante comunidad y hacemos evolucionar la plataforma para satisfacer sus necesidades.

En esencia, la plataforma GitHub Enterprise se centra en la experiencia del desarrollador: tiene la escala necesaria para ofrecer perspectivas que cambian el sector, capacidades de colaboración para una eficiencia transformadora, las herramientas para aumentar la productividad, seguridad en cada paso y la inteligencia artificial para impulsarlo todo a nuevas cotas en una única plataforma integrada.

Introducción a Repositorios

¿Qué es un repositorio? Creación de un repositorio ¿Qué son los gists? ¿Qué son las wikis?

Componentes del flujo de GitHub

Ramas Confirmaciones Sin modificar: se realiza un seguimiento del archivo, pero no se ha modificado desde la última confirmación. Modificado: el archivo se ha cambiado desde la última confirmación, pero estos cambios aún no están almacenados provisionalmente para la siguiente confirmación. Almacenado provisionalmente: el archivo se ha modificado y los cambios se han agregado al área de almacenamiento provisional (también conocida como índice). Estos cambios están listos para confirmarse. Confirmado: el archivo se encuentra en la base de datos del repositorio. Representa la versión confirmada más reciente del archivo.

Qué son las solicitudes de incorporación de cambios?

El flujo de GitHub

l primer paso del flujo de GitHub consiste en crear una rama para que los cambios, características y correcciones que cree no afecten a la rama principal. El segundo paso es realizar los cambios. Se recomienda implementar cambios en la rama de características antes de combinarlos en la rama principal. De esta forma, se tiene la seguridad de que los cambios son válidos en un entorno de producción. El tercer paso consiste en crear una solicitud de incorporación de cambios para pedir comentarios a los colaboradores. La revisión de solicitude sde cambios es tan valiosa que algunos repositorios requieren una revisión aprobatoria antes de que estas se puedan fusionar. A continuación, el cuarto paso consiste en revisar e implementar los comentarios de los colaboradores. Una vez que se sienta a gusto con los cambios, el quinto paso es aprobar la solicitud de incorporación de cambios y combinarla en la rama principal. El sexto y último paso es eliminar la rama. Al eliminar la rama se indica que el trabajo en la rama se ha completado y se evita que usted u otros usuarios empleen accidentalmente ramas antiguas.

GitHub es una plataforma colaborativa

Incidencias Debates Habilitación de un debate en el repositorio

Qué son las páginas de GitHub

Para finalizar nuestro recorrido por GitHub, analicemos las páginas de GitHub.

Puede usar páginas de GitHub para publicitar y hospedar un sitio web sobre usted, su organización o su proyecto directamente desde un repositorio de GitHub.com.

GitHub Pages es un servicio de hospedaje de sitios estáticos que toma archivos HTML, CSS y JavaScript directamente desde un repositorio de GitHub. Opcionalmente, puede ejecutar los archivos a través de un proceso de compilación y publicar un sitio web.

Simplemente edite e introduzca los cambios y ya está, su proyecto estará disponible para el público de una manera visualmente organizada.

A continuación, le guiaremos por un ejercicio para empezar a trabajar con GitHub.

Podrá:

Creación de un nuevo repositorio Creación de una rama Confirmar un archivo Apertura de una solicitud de incorporación de cambios Y combinar una solicitud de cambios

7. Código con GitHub Codespaces

GitHub Codespaces es un entorno de desarrollo totalmente configurado que se hospeda en la nube. Con GitHub Codespaces, el área de trabajo está disponible desde cualquier equipo con acceso a Internet, junto con todos los entornos de desarrollo configurados.

Introducción

Podemos pensar en ello como nuestro *taller de desarrollo portátil*. Todo lo que se necesita para construir algo increíble, pero no está limitado a una ubicación específica. Es como llevar tu taller de desarrollo en la nube contigo a donde quieras. Esto significa que no tienes que perder tiempo configurando todo cada vez que cambias de computadora. GitHub Codespaces ya tiene todo preparado para que puedas concentrarte en lo que realmente importa: escribir código y construir cosas increíbles.

7.1. Ciclo de vida de un codespace

El ciclo de vida de un Codespace tiene varias etapas que hacen que trabajar en proyectos sea más fluido. Al crear un Codespace, estamos en la fase de *Inicio*. Es como abrir nuestro espacio de desarrollo en la nube, listo para trabajar.

Luego viene la fase de *Desarrollo*. Aquí es donde realmente hacemos nuestra magia: escribimos código, realizamos cambios y hacemos todo lo necesario para que nuestro proyecto cobre vida.

Cuando terminamos de trabajar, entramos en la fase de *Suspensión*. Es como cerrar temporalmente nuestro espacio de desarrollo. La próxima vez que lo necesitemos, simplemente lo *Reactivamos* y volvemos a donde lo dejamos, como si nunca nos hubiéramos ido.

Finalmente, cuando ya no necesitamos el Codespace, llegamos a la fase de *Eliminación*. Es como cerrar definitivamente nuestro taller de desarrollo en la nube. Esta estructura cíclica facilita la gestión de nuestros entornos de desarrollo, adaptándose a nuestras necesidades a lo largo del tiempo.

7.1.1. Crear un codespace

Puede crear un codespace en GitHub.com, en Visual Studio Code o en la CLI de GitHub. Existen cuatro formas de crear un codespace:

- Desde una plantilla de GitHub o desde cualquier repositorio de plantillas de GitHub.com para iniciar un nuevo proyecto.

- Desde una rama del repositorio para el trabajo de nuevas características.
- Desde una solicitud de cambios abierta para explorar el trabajo en curso.
- Desde una confirmación en el historial de un repositorio para investigar un error en un punto específico del tiempo.

Puedes usar un Codespace temporalmente para realizar pruebas de código o regresar al mismo Codespace para trabajar en características a largo plazo. Tienes la flexibilidad de crear varios Codespaces por repositorio o incluso por rama. Sin embargo, existe un límite en la cantidad de Codespaces que puedes crear y ejecutar simultáneamente. Si alcanzas este límite, deberás quitar o eliminar un Codespace existente antes de crear uno nuevo.

Además, puedes decidir entre crear un nuevo Codespace cada vez que uses GitHub Codespaces o mantener uno a largo plazo para una característica específica. Si estás comenzando un proyecto nuevo, puedes crear un Codespace a partir de una plantilla y luego publicarlo en un repositorio de GitHub cuando estés listo. Esto brinda flexibilidad según tus necesidades y facilita la gestión de tus entornos de desarrollo en GitHub Codespaces.

Proceso de creación de un codespace

El proceso de creación de un Codespace es sencillo y proporciona un entorno de desarrollo listo para usar. Primero, seleccionas el botón Çode.[en] tu repositorio de GitHub y eliges .ºpen with Codespaces". GitHub automáticamente crea un entorno de desarrollo en la nube basado en la configuración de tu repositorio.

Por ejemplo, si estás trabajando en un proyecto web, al abrir el Codespace, tendrás acceso a un entorno con las versiones correctas del lenguaje de programación, las herramientas y las dependencias que necesitas. Puedes comenzar a codificar de inmediato, sin preocuparte por la instalación o configuración del entorno.

Este proceso es particularmente útil para probar cambios rápidos, colaborar en proyectos o simplemente para tener un entorno de desarrollo consistente sin importar la máquina que estés utilizando. Es una forma eficiente y conveniente de iniciar el desarrollo de proyectos en minutos.

Pasos para la creación

1. **Navegar al Repositorio:**
   - Accede al repositorio en GitHub donde deseas crear el Codespace.
2. **Seleccionar Code y Open with Codespaces:**
   - Haz clic en el botón Çode.[en] la página del repositorio.
   - Selecciona .ºpen with Codespaces.[en] el menú desplegable.
3. **Esperar a la Creación del Codespace:**
   - GitHub iniciará el proceso de creación del Codespace. Este puede llevar unos minutos, dependiendo de la configuración de tu proyecto
4. **Acceder al Codespace:**

- Una vez creado, puedes acceder al Codespace directamente desde tu navegador web. GitHub te proporcionará un entorno de desarrollo totalmente funcional basado en la configuración de tu repositorio.

5. **Explorar y Codificar:**
   - Dentro del Codespace, tendrás acceso a un entorno que incluye el código del repositorio, las dependencias, las extensiones y las configuraciones necesarias para trabajar en tu proyecto
6. **Realizar Cambios y Guardar:**
   - Puedes comenzar a realizar cambios en el código directamente desde el entorno del Codespace. Los cambios se guardan automáticamente y puedes interactuar con el repositorio de GitHub como lo harías normalmente.
7. **Cierre y Persistencia (Opcional):**
   - Si cierras la ventana del navegador o decides no usar el Codespace por un tiempo, GitHub guardará el estado actual. Puedes reanudar tu trabajo más tarde y continuar exactamente desde donde lo dejaste

7.2. Personalizar su codespace

GitHub Codespaces es un entorno dedicado. Puede configurar los repositorios con un contenedor de desarrollo para definir su entorno predeterminado de GitHub Codespaces y personalizar la experiencia de desarrollo en todos los codespaces con dotfiles y Sincronización de configuración.

Qué se puede personalizar

- Sincronización de configuración: puede sincronizar la configuración de Visual Studio Code (VS Code) entre la aplicación de escritorio y el cliente web de VS Code.
- Dotfiles: puede usar un repositorio dotfiles para especificar scripts, preferencias del shell y otras configuraciones.
- Cambiar un codespace de nombre: al crear un codespace, se le asigna un nombre para mostrar generado automáticamente. Si tiene varios codespaces, el nombre para mostrar le ayuda a diferenciar entre ellos. Puede cambiar el nombre para mostrar del codespace.
- Cambiar el shell: puede cambiar el shell en un codespace para mantener su configuración habitual. Al trabajar en un codespace, puede abrir una nueva ventana de terminal con un shell de su elección, cambiar el shell predeterminado para las nuevas ventanas de terminal o instalar un nuevo shell. También puedes usar dotfiles para configurar el shell.
- Cambiar el tipo de máquina: puede cambiar el tipo de máquina que ejecuta el codespace para usar los recursos adecuados para el trabajo que lleva a cabo.
- Establecer el editor predeterminado: puede establecer el editor predeterminado para codespaces en su página de configuración personal. Establezca el editor de su preferencia para que, al crear un codespace o abrir un codespace existente, se abra en el editor predeterminado.
  - Visual Studio Code (aplicación de escritorio)

  - Visual Studio Code (aplicación cliente web)
  - JetBrains Gateway: para abrir codespaces en un IDE de JetBrains
  - JupyterLab (interfaz web para Project Jupyter)

- Establecer la región predeterminada: puede configurar la región predeterminada en la página de configuración de perfil de GitHub Codespaces para personalizar el lugar donde se conservan sus datos.
- Establecer el tiempo de espera: un codespace dejará de ejecutarse después de un período de inactividad. De manera predeterminada, este período es de 30 minutos, pero puede especificar un período de tiempo de espera predeterminado más largo o más corto en su configuración personal en GitHub. La configuración actualizada se aplica a los codespaces que cree o a los ya existentes la próxima vez que los inicie.
- Configuración de eliminación automática: los codespaces inactivos se eliminan de forma automática. Puede elegir cuánto tiempo se conservan los codespaces detenidos, hasta un máximo de 30 días.

7.3. Codespaces versus GitHub.dev editor

Característica	Codespaces	GitHub.dev Editor
Creación de Entornos	En la nube, totalmente configurados	En el navegador, ligero
Accesibilidad	Desde cualquier lugar con conexión a Internet	En el navegador, sin instalación adicional
Configuración	Basada en el repositorio	No es persistente, se reinicia al cerrar la pestaña
Flexibilidad	Varias configuraciones por repositorio o rama	Limitado a un único proyecto por pestaña
Persistencia	Guarda el estado actual para reanudar más tarde	No persiste al cerrar la pestaña
Coste	Cuota mensual gratuita de uso para cuentas personales	Gratuito
Disponibilidad	Disponible para todos en GitHub.com	Disponible para todos en GitHub.com
Extensiones	Con GitHub Codespaces se pueden usar la mayoría de las extensiones de Visual Studio Code Marketplace.	En la vista de extensiones solo aparece un subconjunto de las extensiones que pueden ejecutarse en la web y que se pueden instalar.

Cuadro 1: Comparación entre Codespaces y GitHub.dev Editor.

8. GitHub Copilot

8.1. Introducción a GitHub Copilot

GitHub Copilot es una herramienta revolucionaria respaldada por inteligencia artificial, diseñada para hacer que la escritura de código sea más eficiente y menos laboriosa. Al aprovechar un modelo de lenguaje preentrenado generativo desarrollado por OpenAI Codex, esta innovadora herramienta extrae información del contexto de comentarios y código existente para ofrecer sugerencias instantáneas de líneas de código y funciones completas.

Lo que hace que GitHub Copilot sea excepcional es su capacidad para acelerar el proceso de codificación, permitiéndote abordar problemas de manera más rápida. Además, la investigación ha demostrado que esta herramienta no solo aumenta la velocidad de codificación, sino que también permite a los desarrolladores concentrarse en resolver problemas más sustanciales, mantenerse informados durante más tiempo y experimentar una mayor satisfacción en su trabajo.

Para facilitar su integración en el flujo de trabajo de los desarrolladores, GitHub Copilot ofrece extensiones para populares entornos de desarrollo, como Visual Studio Code, Visual Studio, Neovim y los entornos de desarrollo integrados (IDE) de JetBrains. En resumen, GitHub Copilot representa un hito en el desarrollo de software al combinar la inteligencia artificial con la escritura de código para mejorar la productividad y la experiencia de programación.

8.2. El programador de pares de IA

GitHub Copilot es como tener a un compañero de programación impulsado por inteligencia artificial a mi disposición. Esta innovadora herramienta transforma la forma en que desarrollo código al sugerir líneas individuales y funciones completas en tiempo real, todo basado en el contexto de mis comentarios y código existente. No solo acelera el proceso de codificación, sino que también ha demostrado aumentar significativamente mi productividad según investigaciones de GitHub y Microsoft.

Lo asombroso de GitHub Copilot es su capacidad para trabajar con una amplia variedad de lenguajes de programación conocidos, brindándome asistencia independientemente del proyecto en el que esté trabajando. Desde su lanzamiento, ha demostrado ser un recurso valioso para los desarrolladores, proporcionando sugerencias precisas y mejorando mi flujo de trabajo de manera tangible. En resumen, GitHub Copilot es más que una herramienta, es como tener a un inteligente compañero de codificación a mi lado, listo para ayudarme a escribir código de manera rápida y eficiente.

8.3. ChatGPT en el editor con GitHub Copilot Chat

Copilot X aporta una interfaz de chat al editor que se centra en escenarios de desarrollador y se integra de forma nativa con VS Code y Visual Studio. Reconoce qué código ha escrito un desarrollador, qué mensajes de error se muestran y está profundamente insertado en el IDE. Un desarrollador puede obtener análisis detallados y explicaciones de para qué están diseñados los bloques de código, generar pruebas unitarias e incluso obtener correcciones propuestas para errores.

8.4. Configuración a GitHub Copilot

8.5. Registro en GitHub Copilot

8.6. GitHub Copilot en Visual Studio Code

8.7. Uso de GitHub Copilot con JavaScript

8.8. Uso de GitHub Copilot con Python

9. Proyecto de desafío 01

Crear una aplicación de consola de minijuegos con GitHub Copilot

9.1. Introducción

La creación de un minijuego puede ayudarle a practicar sus habilidades de programación y mejorar su capacidad para crear aplicaciones de consola en Python.

En este módulo, desarrollará el minijuego clásico de piedra, papel, tijeras con la ayuda de GitHub Codespaces y GitHub Copilot. Es decir, no es necesario preocuparse por configurar el entorno de desarrollo, por lo que puede centrarse en el desarrollo de aplicaciones mientras se basa en un código asistente.

9.2. Ejercicio: Crear la lógica del juego

Especificación

- Reglas del juego:
  - La piedra gana a las tijeras (las rompe).
  - Las tijeras han ganado al papel (lo cortan).
  - El papel gana a la piedra (la envuelve).
  - El minijuego es multijugador y el equipo juega el papel del oponente y elige un elemento aleatorio de la lista de elementos
- Interacción con el jugador:
  - La consola se usa para interactuar con el jugador.
  - El jugador puede elegir una de las tres opciones: roca, papel o tijeras.
  - El jugador puede elegir si vuelve a jugar.
  - Se debe advertir al jugador si introduce una opción no válida.
  - El jugador ve su puntuación al final del juego.
- Validación de la entrada del usuario:
  - En cada ronda, el jugador debe entrar en una de las opciones de la lista y ser informado de si ganó, perdió o empató con el oponente.

- El minijuego debe controlar las entradas del usuario, colocarlas en minúsculas e informar al usuario si la opción no es válida.
- Al final de cada ronda, el jugador debe responder si quiere jugar de nuevo o no.

```
Vamos a crear el juego de piedra papel o tijeras en python. La computadora es
le jugador 1 y el usuario es el jugador 2
```

```
Vamos a crear el juego de piedra papel o tijeras en python. La computadora es
le jugador 1 y el usuario es el jugador 2
```

10. Proyecto de desafío 02

Agregar funcionalidades de generación y análisis de imágenes a la aplicación

10.1. Introducción

Computer Vision es un área principal de inteligencia artificial (IA), destinada a procesar y analizar la información contenida en orígenes de datos visuales, como imágenes y vídeos, para extraer información útil. Esta rama de IA emula las funcionalidades visuales humanas mediante algoritmos avanzados: las tecnologías de inteligencia artificial de vanguardia han alcanzado un nivel de precisión comparable a la visión humana en muchas tareas, como la clasificación de imágenes, la detección de objetos y la descripción de imágenes.

Saber cómo integrar Computer Vision en sus aplicaciones es fundamental para desarrollar soluciones modernas e inteligentes en varios dominios, como el comercio minorista o la seguridad, y para garantizar la accesibilidad a los servicios, por ejemplo, proporcionando una descripción de texto de las imágenes que comparte en su sitio web.

10.2. Azure

Azure es una plataforma de computación en la nube de Microsoft que ofrece una amplia gama de servicios, desde infraestructura como servicio (IaaS) hasta plataforma como servicio (PaaS) y software como servicio (SaaS). Azure permite a las empresas de todos los tamaños crear, implementar y administrar aplicaciones en la nube de manera rápida y sencilla.

- IaaS

  Los servicios IaaS de Azure brindan a las empresas la infraestructura de TI básica que necesitan para ejecutar sus aplicaciones, como servidores, almacenamiento y redes. Azure ofrece una variedad de opciones de IaaS para satisfacer las necesidades de las empresas, desde servidores virtuales hasta centros de datos completos.

- PaaS

  Los servicios PaaS de Azure brindan a las empresas un entorno completo para desarrollar, implementar y administrar aplicaciones. Azure ofrece una variedad de servicios PaaS para diferentes tipos de aplicaciones, como aplicaciones web, aplicaciones móviles y aplicaciones de análisis.

- SaaS

  Los servicios SaaS de Azure brindan a las empresas software listo para usar que se ejecuta en la nube. Azure ofrece una variedad de servicios SaaS para diferentes necesidades comerciales, como correo electrónico, colaboración, análisis y aprendizaje automático.

Algunos de los beneficios de usar Azure incluyen:

- Flexibilidad: Azure ofrece una amplia gama de servicios que las empresas pueden usar para satisfacer sus necesidades específicas.
- Escalabilidad: Azure es escalable, lo que significa que las empresas pueden aumentar o disminuir sus recursos según sea necesario.
- Seguridad: Azure ofrece una variedad de funciones de seguridad para proteger los datos y las aplicaciones de las empresas.

10.3. Azure Static Web App

Azure Static Web Apps es un servicio de Azure que permite a los desarrolladores crear, implementar y administrar aplicaciones web estáticas en la nube. Las aplicaciones web estáticas son aquellas que no requieren un servidor web para ejecutarse. En su lugar, se componen de archivos HTML, CSS y JavaScript que se pueden servir directamente desde un servidor de archivos.

Para crear una aplicación web estática con Azure Static Web Apps, solo necesitas tener un repositorio de código Git con tu aplicación. Luego, puedes usar la CLI de Azure o el portal de Azure para crear una nueva aplicación web estática y conectarla a tu repositorio de código.

Una vez que tu aplicación web estática esté creada, Azure la construirá y la implementará en la nube. Tu aplicación estará disponible en una URL global que puedes compartir con tus usuarios.

10.4. Azure OpenAI

Proporciona acceso a través de la API REST a potentes modelos de lenguaje, como GPT-4, GPT-3.5-Turbo e Embeddings de OpenAI. Las series de modelos GPT-4 y GPT-3.5-Turbo ya están disponibles de forma generalizada. Lo genial es que estos modelos se pueden adaptar fácilmente a tareas específicas, como generación de contenido, resumen, búsqueda semántica y traducción de lenguaje natural a código. Puedes acceder al servicio a través de API REST, el SDK de Python o la interfaz web en Azure OpenAI Studio.

10.5. Seguridad de los servicios de Azure AI

La seguridad debe considerarse una prioridad máxima en el desarrollo de toda aplicación y, con el crecimiento de las aplicaciones compatibles con la inteligencia artificial, la seguridad es aún más importante. En este artículo se describen varias características de seguridad disponibles para los servicios de Azure AI. Cada característica aborda una responsabilidad específica, por lo que se pueden usar varias características en el mismo flujo de trabajo.

10.6. React

React es una biblioteca de JavaScript de código abierto que se utiliza para crear interfaces de usuario. Es una de las bibliotecas de JavaScript más populares y se utiliza para crear una amplia gama de aplicaciones web, móviles y de escritorio.

React se basa en el concepto de componentes, que son piezas reutilizables de código que pueden combinarse para crear interfaces de usuario complejas. Los componentes son fáciles de entender y mantener, lo que hace que React sea una buena opción para proyectos de cualquier tamaño.

React se puede utilizar para crear una amplia gama de interfaces de usuario, desde aplicaciones web simples hasta aplicaciones web complejas. Algunas de las cosas que se pueden hacer con React incluyen:

- Crear aplicaciones web interactivas: React es ideal para crear aplicaciones web interactivas, como aplicaciones de comercio electrónico, aplicaciones de redes sociales y juegos.
- Crear aplicaciones web optimizadas para dispositivos móviles: React se puede utilizar para crear aplicaciones web optimizadas para dispositivos móviles, que se ven y funcionan bien en teléfonos inteligentes y tabletas.
- Crear aplicaciones web de una sola página: React se puede utilizar para crear aplicaciones web de una sola página, que no requieren recargas de página para actualizar el contenido.

React tiene una serie de características que lo hacen una biblioteca de JavaScript potente y versátil. Estas características incluyen:

- Componentes: React se basa en componentes, que son bloques de construcción reutilizables que se pueden combinar para crear interfaces de usuario complejas.
- DOM virtual: React utiliza un DOM virtual para representar el estado de una interfaz de usuario. Esto permite que React sea eficiente y reactivo.
- Hooks: Los hooks son funciones que permiten a los desarrolladores acceder a los estados y propiedades de los componentes sin tener que escribir clases.

Ventajas de React

React ofrece una serie de ventajas sobre otras bibliotecas de JavaScript para crear interfaces de usuario. Estas ventajas incluyen:

- Facilidad de uso: React es relativamente fácil de aprender y usar, incluso para desarrolladores principiantes.
- Eficiencia: React es una biblioteca eficiente que puede manejar grandes cantidades de datos y eventos.
- Reactividad: React es una biblioteca reactiva que puede actualizar las interfaces de usuario en respuesta a los cambios de datos.
- Escalabilidad: React es una biblioteca escalable que puede manejar aplicaciones web complejas.