KAREN GUADALUPE MORENO HERNÁNDEZ 22121036
04 AGOSTO 2024

UNIDAD 2 TAREA. PROGRAMACIÓN AVANZADA

1. DEFINICIONES Y CONCEPTOS

1.1 Control de versiones

- **Definición**: Un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, permitiendo que versiones específicas puedan ser recuperadas más tarde.
- **Explicación**: El control de versiones es esencial en el desarrollo de software, ya que permite a los equipos trabajar de manera colaborativa sin sobrescribir el trabajo de otros. Git es un ejemplo de sistema de control de versiones distribuido.

1.2 Repositorio local

- **Definición**: Es una copia del proyecto almacenada en la computadora del usuario, donde se realizan los cambios y se gestionan antes de ser compartidos.
- **Explicación**: Los repositorios locales permiten a los desarrolladores trabajar en sus proyectos sin necesidad de estar conectados a internet. Los cambios se pueden realizar, confirmar y luego subir a un repositorio remoto.

1.3 Repositorio remoto

- **Definición**: Es una versión del repositorio local que se aloja en un servidor, como GitHub, y que otros desarrolladores pueden clonar y colaborar en él.
- **Explicación**: El repositorio remoto permite que los cambios realizados localmente se sincronicen y compartan con otros miembros del equipo.

1.4 Commit

- **Definición**: Una confirmación de los cambios realizados en el repositorio local, creando un registro en el historial de versiones.
- **Explicación**: Un commit es como un "checkpoint" en el desarrollo, donde se guarda el estado del proyecto en ese momento con un mensaje descriptivo.

1.5 Branch (rama)

- **Definición**: Una bifurcación de la línea de desarrollo principal (master) que permite trabajar en nuevas características o corregir errores sin afectar la línea principal.
- **Explicación**: Las ramas son útiles para desarrollar nuevas funcionalidades de manera aislada, y luego integrar esos cambios de nuevo en la rama principal cuando estén listos.

1.6 Merge (combinar)

- **Definición**: El proceso de integrar los cambios de una rama en otra, usualmente la rama principal.
- **Explicación**: Merging se utiliza para traer los cambios realizados en una rama separada de vuelta a la línea de desarrollo principal, combinando las versiones.

1.7 Pull request

- **Definición**: Una solicitud para revisar y posiblemente fusionar los cambios de una rama en el repositorio principal.
- **Explicación**: Los pull requests son esenciales en proyectos colaborativos, donde otros miembros del equipo revisan y aprueban los cambios antes de integrarlos.

1.8 Fork (bifurcación)

- **Definición**: Una copia de un repositorio que permite realizar cambios en un proyecto sin afectar el repositorio original.
- **Explicación**: Forking es común en proyectos de código abierto, donde se desea modificar o mejorar un proyecto sin interferir en el desarrollo del original.

1.9 Clone (clonar)

- **Definición**: El acto de descargar una copia completa de un repositorio remoto a un repositorio local.
- **Explicación**: Clonar un repositorio permite a los desarrolladores tener una copia exacta del proyecto en su máquina local para trabajar en él.

1.10 Conflictos de merge

- **Definición**: Situaciones donde Git no puede combinar automáticamente dos conjuntos de cambios debido a que los mismos archivos fueron modificados de manera incompatible.
- **Explicación**: Los conflictos de merge deben resolverse manualmente para decidir qué cambios conservar antes de completar el proceso de fusión.

1.11 Gitignore

- **Definición**: Un archivo que especifica qué archivos o directorios deben ser ignorados por Git al realizar commits.
- **Explicación**: gitignore es útil para evitar incluir archivos innecesarios o sensibles en el repositorio, como configuraciones locales o archivos temporales.

2. COMPARACIÓN ENTRE GIT Y GITHUB

2.1 Funciones principales

- **Git**: Es un sistema de control de versiones distribuido que permite a los desarrolladores gestionar el historial de versiones de un proyecto localmente.
- **GitHub**: Es una plataforma de alojamiento de código que utiliza Git como base y agrega funcionalidades colaborativas como pull requests, revisiones de código, y manejo de issues.

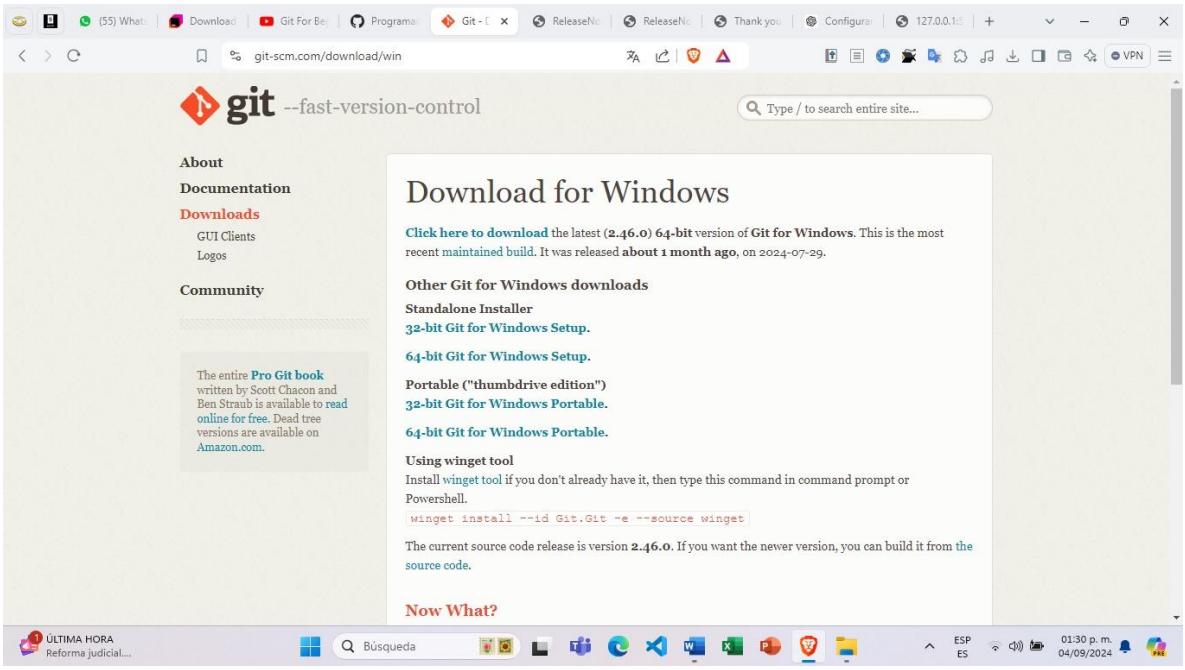
2.2 Relación entre Git y GitHub

- **Relación**: Git se utiliza para gestionar los cambios de un proyecto localmente, mientras que GitHub actúa como el repositorio remoto donde se almacenan y comparten esos cambios con otros colaboradores.

3. INSTALACIÓN Y CONFIGURACIÓN DE GIT

3.1 Instrucciones paso a paso para instalar Git en diferentes sistemas operativos:

- **Windows**:
    1. Descarga Git desde el sitio oficial.
    2. Ejecuta el instalador y sigue las instrucciones.
    3. Configura Git usando los siguientes comandos en la terminal:



bash

Copiar código

git config --global user.name "Tu Nombre"

git config --global user.email "tuemail@ejemplo.com"

- **macOS**:
    1. Abre la Terminal.
    2. Instala Git usando Homebrew:

bash

Copiar código

brew install git

    3. Configura Git con los comandos mencionados anteriormente.

- **Linux**:
    1. Abre la Terminal.
    2. Instala Git usando el gestor de paquetes de tu distribución, por ejemplo:

bash

Copiar código

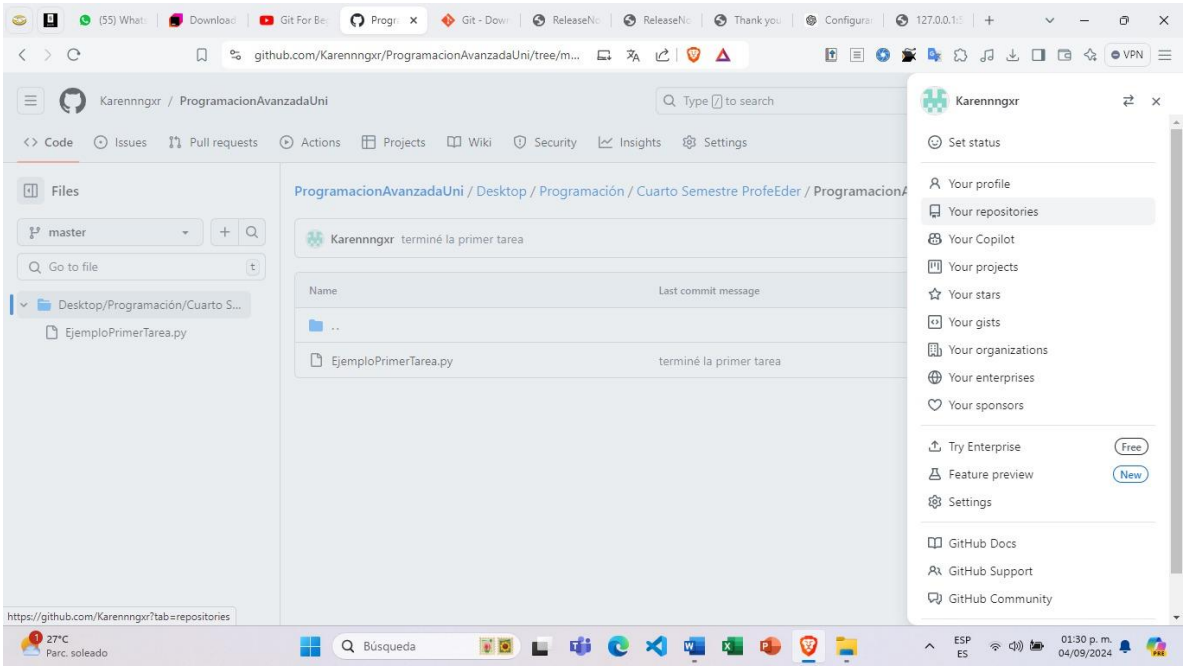
sudo apt-get install git

    3. Configura Git con los comandos mencionados anteriormente.

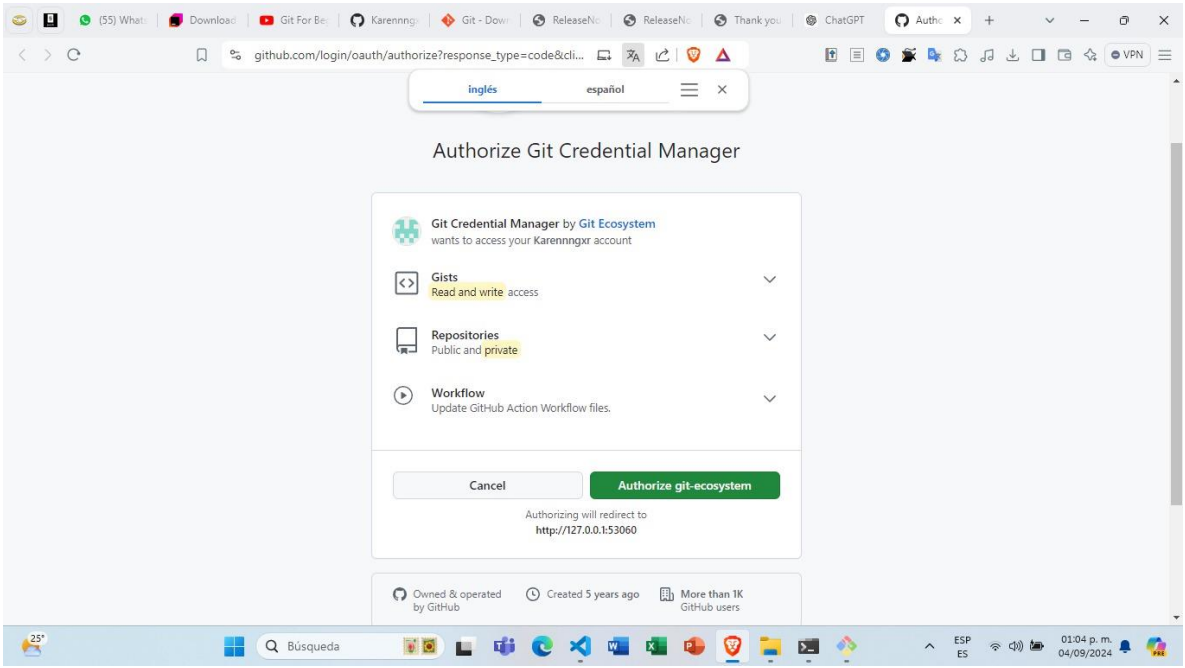
4. CREACIÓN Y GESTIÓN DE REPOSITORIOS EN GITHUB

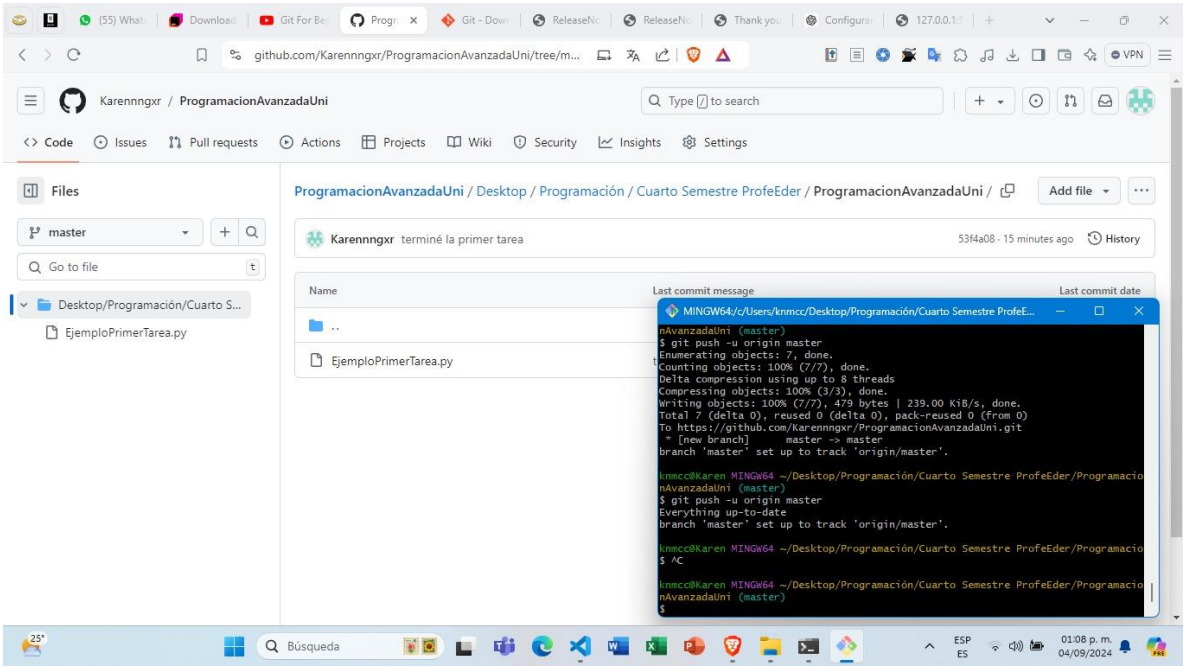
4.1 Crear un nuevo repositorio en GitHub:

1. Inicia sesión en GitHub.
2. Haz clic en "New repository".
3. Rellena los detalles como nombre del repositorio, descripción, y visibilidad.
4. Selecciona opciones como Initialize this repository with a README si deseas comenzar con un archivo README.
5. Haz clic en "Create repository".



Para utlizar nuestro repertorio tambien descargamos VistualStudio Code y Intellij IDEA, en este ejemplo solo usamos VisualStudio para crear un ejemplo de programa en python, y junto con Git lo transladamos con comandos a GitHub como se muestra en las capturas.





Resumen paso a paso de los comandos que utilizaste para vincular Git con GitHub, junto con los errores que encontraste y cómo los resolvimos.

Paso 1: Inicializar un repositorio en Git localmente

1. **Comando**:

bash

Copiar código

git init

   - Este comando inicializa un nuevo repositorio Git en tu directorio actual.

2. **Resultado esperado**:

   - Git crea un repositorio vacío en la carpeta donde ejecutaste el comando.

Paso 2: Agregar archivos al área de preparación (staging area)

1. **Comando**:

bash

Copiar código

git add .

- Este comando agrega todos los archivos en el directorio actual al área de preparación.

2. **Resultado esperado**:
    - Los archivos están listos para ser confirmados (commiteados) en el repositorio local.

Paso 3: Crear un commit

1. **Comando**:

bash

Copiar código

git commit -m "Initial commit"

- Este comando guarda los cambios en el repositorio local con un mensaje de confirmación.

2. **Resultado esperado**:
    - Se crea un commit con el mensaje "Initial commit".

Paso 4: Vincular el repositorio local con GitHub

1. **Primera tentativa**:

Comando:

bash

Copiar código

git remote add origin https://github.com/Karennngxr/ProgramacionAvanzadaUni.git

Error posible:

- Si este comando arroja un error diciendo que remote origin already exists, significa que ya has vinculado un remoto con ese nombre. En ese caso, puedes eliminar el origen antiguo o sobrescribirlo.

Solución:

- Para sobrescribir el origen existente:

bash

Copiar código

git remote set-url origin https://github.com/Karennngxr/ProgramacionAvanzadaUni.git

- Si prefieres eliminar el origen anterior y luego añadir el nuevo:

bash

Copiar código

git remote remove origin

git remote add origin https://github.com/Karennngxr/ProgramacionAvanzadaUni.git

2. **Resultado esperado**:
    - Ahora, el repositorio local está vinculado con el repositorio remoto en GitHub.

Paso 5: Empujar los cambios al repositorio remoto en GitHub

1. **Primera tentativa**:

Comando:

bash

Copiar código

git push -u origin master

Error posible:

- Podría aparecer un mensaje solicitando autenticación. Asegúrate de usar las credenciales correctas (nombre de usuario y token personal de acceso en lugar de contraseña).

Solución:

- Si aparece una ventana emergente solicitando autenticación, puedes iniciar sesión a través de tu navegador o introducir un código. Si Git te pide credenciales, asegúrate de haber configurado tu token de acceso personal en GitHub para reemplazar la contraseña.

2. **Resultado esperado**:
    - Se suben los archivos y commits al repositorio remoto en la rama master.

3. **Verificación**:
    - Si intentas empujar de nuevo con:

bash

Copiar código

git push -u origin master

y recibes el mensaje Everything up-to-date, significa que todos los cambios locales ya están sincronizados con el repositorio remoto.

Paso 6: Verificar que todo está vinculado correctamente

1. **Comando**:

bash

Copiar código

git status

    - Verifica si hay archivos que aún no se han subido al repositorio remoto.
2. **Resultado esperado**:
    - El mensaje On branch master, Your branch is up to date with 'origin/master' indica que no hay cambios pendientes de ser empujados.

4.2 Gestión del repositorio:

- **Colaboradores**: Puedes añadir colaboradores desde la pestaña "Settings" en la sección "Collaborators".
- **Configuraciones**: Desde la pestaña "Settings" puedes configurar opciones avanzadas como Webhooks, Integrations, y más.

5. COLABORACIÓN Y FLUJO DE TRABAJO EN GITHUB

5.1 Propósito de cada paso en el flujo de trabajo:

- **Branches**: Facilitan el trabajo en nuevas funcionalidades sin afectar el código base.
- **Commits**: Permiten guardar el progreso con mensajes descriptivos.
- **Merges**: Integran los cambios de diferentes ramas en la principal.

- **Pull Requests**: Facilitan la revisión de código y la colaboración antes de fusionar ramas.

6. CASOS DE USO Y EJEMPLOS PRÁCTICOS

6.1 Ejemplos en la industria:

- **Desarrollo de Software Colaborativo**: Empresas como Microsoft utilizan Git y GitHub para gestionar grandes proyectos con múltiples desarrolladores trabajando en paralelo.
- **Proyectos de Código Abierto**: Ejemplos como Linux Kernel, donde miles de contribuyentes pueden colaborar en un solo proyecto utilizando forks, pull requests y branches.

7. DESAFÍO ADICIONAL

7.1 Funcionalidad avanzada de Git o GitHub:

- **Rebase**: Es una funcionalidad que permite aplicar commits de una rama sobre otra, reescribiendo el historial de commits para crear un historial más limpio.
- **Beneficios**: Rebase ayuda a mantener un historial de commits lineal y más fácil de seguir, especialmente en proyectos grandes con muchos colaboradores.