Introducción a GitHub

MSc. Ing. Oswaldo Espinoza-Hurtado

23 de mayo del 2024

1. Introducción

En esta clase exploraremos las herramientas y prácticas fundamentales para el control de versiones mediante Git y la colaboración en proyectos de software usando GitHub. El control de versiones es esencial en el desarrollo moderno de software, permitiendo a los equipos trabajar de manera eficiente y organizada. Git, un sistema de control de versiones distribuido, junto con GitHub, una plataforma de alojamiento de código basada en la nube, nos proporcionarán las bases necesarias para gestionar nuestro código de manera efectiva, facilitando la colaboración y asegurando la integridad de nuestros proyectos.

1.1. ¿Qué es Git?

Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear los cambios en el código fuente a lo largo del tiempo. Fue creado por Linus Torvalds en 2005 para gestionar el desarrollo del núcleo de Linux. Git proporciona una estructura para la gestión de versiones, permitiendo la creación de múltiples ramas, la fusión de cambios y la reversión a estados anteriores del código. Es altamente eficiente y escalable, utilizado tanto por proyectos pequeños como por grandes empresas tecnológicas en todo el mundo.

1.2. ¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo basada en la nube que utiliza Git como sistema de control de versiones. Fundada en 2008, GitHub facilita la gestión de repositorios de código, ofreciendo herramientas adicionales como gestión de proyectos, seguimiento de problemas, revisión de código y acciones automatizadas para integración continua y despliegue continuo (CI/CD). GitHub no solo permite alojar repositorios, sino también colaborar con otros desarrolladores de manera eficiente, compartiendo y revisando código en tiempo real.

1.3. Diferencias entre Git y GitHub

Git es una herramienta de control de versiones que se instala y se ejecuta localmente en las máquinas de los desarrolladores, permitiendo gestionar y rastrear los cambios en los archivos de un proyecto. Por otro lado, GitHub es un servicio de alojamiento de repositorios Git que proporciona una interfaz gráfica

y herramientas adicionales para facilitar la colaboración y la gestión de proyectos en línea. Mientras que Git se encarga de las operaciones de versiones a nivel local, GitHub centraliza la colaboración y el acceso remoto a los repositorios.

1.4. Importancia del Control de Versiones en el Desarrollo de Software y en la Gestión de Bases de Datos

El control de versiones es crucial en el desarrollo de software porque permite a los equipos trabajar de manera colaborativa y organizada, gestionando los cambios en el código de forma segura y eficiente. Con Git, los desarrolladores pueden trabajar en diferentes partes del proyecto simultáneamente sin riesgo de sobrescribir el trabajo de los demás, facilitando la integración de los cambios y la resolución de conflictos. Además, en la gestión de bases de datos, el control de versiones permite rastrear cambios en los esquemas y en los datos, asegurando la coherencia y facilitando la recuperación ante fallos. GitHub amplifica estos beneficios al proporcionar un entorno centralizado y colaborativo donde los equipos pueden gestionar sus proyectos, compartir conocimientos y automatizar flujos de trabajo.

2. Instalación de Git

Para comenzar a utilizar Git, es necesario instalarlo en tu sistema. Vamos a utilizar la terminal de RStudio para llevar a cabo la instalación. En sistemas Windows, puedes instalar Git utilizando el siguiente comando en la terminal de RStudio:

```
winget install --id Git.Git -e --source winget
```

Este comando descargará e instalará Git desde la fuente oficial utilizando el administrador de paquetes winget.

Configuración de Git (nombre de usuario, correo electrónico) Una vez que Git esté instalado, es importante configurarlo con tu nombre de usuario y correo electrónico. Esta configuración es necesaria para que Git pueda asociar los cambios que realices con tu identidad. Utiliza los siguientes comandos en la terminal de RStudio:

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu.correo@ejemplo.com"
```

Estos comandos configuran tu nombre de usuario y correo electrónico a nivel global, es decir, se aplicarán a todos los repositorios en los que trabajes desde este equipo.

3. Comandos Básicos de Git

3.1. git init

El comando **git init** se utiliza para inicializar un nuevo repositorio de Git. Crea un nuevo subdirectorio .git en el directorio de trabajo, que contiene todos

los archivos necesarios del repositorio, como la configuración y el historial de versiones.

```
git init
```

3.2. git clone

El comando git clone se utiliza para clonar un repositorio existente. Este comando crea una copia del repositorio especificado en tu máquina local.

```
git clone https://github.com/usuario/repositorio.git
```

3.3. git status

El comando git status muestra el estado del repositorio, incluyendo los archivos que han cambiado, los archivos que están en el área de preparación y los archivos que no están siendo rastreados por Git.

```
git status
```

3.4. git add

El comando git add se utiliza para añadir archivos al área de preparación (staging area). Esto prepara los archivos para ser incluidos en el próximo commit.

```
git add archivo.txt
```

3.5. git rm

El comando git rm se utiliza para eliminar archivos del repositorio y del sistema de archivos.

```
git rm archivo.txt
```

3.6. git commit

El comando git commit guarda los cambios en el historial de versiones del repositorio. Es necesario proporcionar un mensaje que describa los cambios realizados.

```
git commit -m "Descripción de los cambios"
```

3.7. git push

El comando git push se utiliza para enviar los cambios confirmados (commits) desde el repositorio local al repositorio remoto.

```
git push origin main
```

3.8. git pull

El comando git pull se utiliza para traer y combinar cambios desde el repositorio remoto al repositorio local.

```
git pull origin main
```

3.9. git branch

El comando git branch se utiliza para listar, crear o eliminar ramas en el repositorio. Las ramas permiten desarrollar características aisladas unas de otras.

```
git branch
git branch nueva-rama
git branch -d nombre-rama
```

3.10. git checkout

El comando git checkout se utiliza para cambiar de una rama a otra o restaurar archivos del historial de versiones.

```
git checkout nombre-rama
```

3.11. git merge

El comando git merge se utiliza para combinar cambios de una rama en la rama actual. Esto es útil para integrar características desarrolladas en ramas separadas.

```
git merge nombre-rama
```

3.12. git log

El comando git log muestra el historial de commits del repositorio, incluyendo mensajes de commit, autores y fechas.

```
git log
```

3.13. git remote set-url origin [nueva URL]

El comando git remote set-url se utiliza para actualizar la URL del repositorio remoto. Esto es útil si la URL del repositorio remoto ha cambiado.

```
git remote set-url origin https://nuevo-repositorio.git
```

4. Creación y Gestión de Repositorios en GitHub

4.1. Crear un Nuevo Repositorio en GitHub

Para crear un nuevo repositorio en GitHub, sigue estos pasos:

1. Inicia sesión en tu cuenta de GitHub.
2. Haz clic en el botón *New* en la esquina superior derecha de la página de inicio.
3. Completa el formulario con el nombre del repositorio y una descripción opcional.
4. Elige si quieres que el repositorio sea público o privado.
5. (Opcional) Añade un archivo README, un archivo .gitignore, y selecciona una licencia.
6. Haz clic en *Create repository* para finalizar.

4.2. Subir un Repositorio Local a GitHub

Para subir un repositorio local a GitHub, utiliza los siguientes comandos en la terminal de RStudio:

1. Inicia un nuevo repositorio

```
git init
```

   o navega a un repositorio existente

```
cd ruta/al/repositorio
```

2. Añade el repositorio remoto de GitHub:

```
git remote add origin https://github.com/usuario/nuevo-repositorio.git
```

3. Añade y confirma tus archivos:

```
git add .
git commit -m "Primer commit"
```

4. Envía tus cambios al repositorio remoto:

```
git push -u origin main
```

4.3. Gestionar Permisos y Colaboradores

Para gestionar permisos y colaboradores en tu repositorio de GitHub:

1. Ve a la página principal del repositorio en GitHub.
2. Haz clic en *Settings* (Configuración).
3. Selecciona *Collaborators* (Colaboradores) en el menú de la izquierda.
4. Introduce el nombre de usuario o correo electrónico del colaborador y haz clic en *Add collaborator* (Añadir colaborador).
5. El colaborador recibirá una invitación para unirse al repositorio.

4.4. Navegación por la Interfaz de GitHub

La interfaz de GitHub ofrece varias herramientas y funcionalidades para gestionar y colaborar en proyectos:

- **Issues (Problemas):** Permiten rastrear errores, tareas y solicitudes de nuevas características. Puedes asignar issues a colaboradores, etiquetarlos y comentarlos.
- **Pull Requests (Solicitudes de Extracción):** Utilizadas para revisar y discutir cambios antes de fusionarlos en la rama principal. Pueden incluir comentarios en línea, revisiones de código y aprobación de cambios.
- **Actions (Acciones):** Permiten configurar flujos de trabajo automatizados para CI/CD. Puedes definir workflows que se ejecutan en respuesta a eventos como commits o pull requests.
- **Wiki:** Un espacio para documentar tu proyecto. Puedes crear y editar páginas directamente desde el navegador.
- **Projects (Proyectos):** Tableros Kanban para gestionar tareas y planificar el trabajo. Puedes crear tarjetas, asignarlas a colaboradores y moverlas entre columnas de acuerdo al estado del trabajo.

5. Colaboración en GitHub

5.1. Forks y Pull Requests

GitHub facilita la colaboración en proyectos mediante el uso de forks y pull requests. Un fork es una copia de un repositorio que se crea en tu cuenta de GitHub, permitiéndote experimentar con cambios sin afectar el proyecto original. Para colaborar en un proyecto, sigue estos pasos:

1. **Fork del repositorio:** Haz clic en el botón *Fork* en la página del repositorio original para crear una copia en tu cuenta de GitHub.
2. **Clonar el fork a tu máquina local:** Utiliza el comando `git clone` para clonar el fork en tu máquina local.

```
git clone https://github.com/tu-usuario/nombre-del-repositorio.git
```

3. **Realizar cambios:** Crea una nueva rama, realiza los cambios necesarios y haz commits.

```
git checkout -b nueva-rama
git add .
git commit -m "Descripción de los cambios"
```

4. **Enviar los cambios a GitHub:** Envía tus cambios al fork en GitHub.

```
git push origin nueva-rama
```

5. **Crear un Pull Request:** En GitHub, abre tu fork, selecciona la rama con los cambios y haz clic en *New Pull Request*. Describe los cambios realizados y envía el pull request para que los mantenedores del proyecto original lo revisen.

5.2. Revisión de Código

La revisión de código es un paso crucial en el flujo de trabajo colaborativo de GitHub. Permite a los colaboradores discutir y mejorar los cambios propuestos antes de fusionarlos en la rama principal. Durante la revisión de código, los revisores pueden:

- **Comentar en líneas específicas del código:** Proporcionar feedback sobre partes específicas del código propuesto.
- **Solicitar cambios:** Indicar al autor del pull request que se requieren ajustes antes de que se pueda fusionar.
- **Aprobar el pull request:** Confirmar que el código está listo para ser fusionado.

El objetivo es asegurar que el código cumple con los estándares del proyecto y que no introduce errores.

5.3. Resolución de Conflictos

Los conflictos de fusión ocurren cuando Git no puede combinar automáticamente los cambios de diferentes ramas. Para resolver conflictos:

1. **Identificar los conflictos:** Git te notificará sobre los conflictos y marcará los archivos afectados.
2. **Resolver los conflictos manualmente:** Edita el contenido para elegir o combinar los cambios deseados.
3. **Marcar los conflictos como resueltos:** Una vez que hayas resuelto los conflictos, añade los archivos modificados al área de preparación.

```
git add archivo-en-conflicto.txt
```

4. **Finalizar la fusión:** Realiza un commit para completar la fusión.

```
git commit -m "Resueltos conflictos de fusión"
```

La colaboración efectiva en GitHub requiere el uso de estas herramientas y técnicas para gestionar los cambios y asegurar la calidad del código.

6. Materiales y Recursos

- **Repositorio de comandos básicos de Git:**
  - ErickChacon/git-commands
- **Lecturas recomendadas:**
  - Pro Git
  - Documentación oficial de GitHub
- **Tutoriales en línea:**
  - GitHub Learning Lab
  - Videos de YouTube sobre Git y GitHub