

Uso de Git y GitHub con Buenas Prácticas y Gestión de Ramas

Objetivo: Aprender a crear y configurar una cuenta en GitHub, crear y administrar repositorios con buenas prácticas, trabajar con tres ramas principales (main, qa, y dev), y colaborar en proyectos utilizando:

fork

clone

pull request

Asegurando que las configuraciones de Git sean locales al repositorio para no afectar otras configuraciones.

Tabla de Contenidos

Creación de una Cuenta y Configuración del Perfil

Creación y Administración de Repositorios en GitHub con Buenas Prácticas

Gestión de Ramas: main, qa, y dev

Clonación Detallada de un Repositorio en Git

Configuración Local de Variables en Git

Colaboración en Proyectos: Fork, Clone, Pull Request

Parte 1: Creación de una Cuenta y Configuración del Perfil

1.1 Crear una Cuenta en GitHub:

Paso 1: Abre tu navegador web y visita GitHub.

Paso 2: Haz clic en el botón "Sign up" o "Registrarse".

Paso 3: Completa los campos requeridos:

Nombre de usuario: Elige un nombre único que te represente.

Correo electrónico: Ingresa una dirección de correo válida.

Contraseña: Crea una contraseña segura.

Paso 4: Sigue las instrucciones en pantalla para completar el registro.

Paso 5: Verifica tu dirección de correo electrónico a través del enlace que recibirás.

1.2 Configurar tu Perfil:

Paso 1: Inicia sesión en GitHub.

Paso 2: Haz clic en tu foto de perfil (esquina superior derecha) y selecciona "Your profile".

Paso 3: Haz clic en "Edit profile".

Paso 4: Completa tu información personal:

Nombre completo

Biografía

Ubicación

Sitio web o blog

Paso 5: Sube una foto de perfil que te identifique.

Paso 6: Personaliza otras configuraciones según tus preferencias.

Parte 2: Creación y Administración de Repositorios en GitHub con Buenas Prácticas

2.1 Crear un Nuevo Repositorio

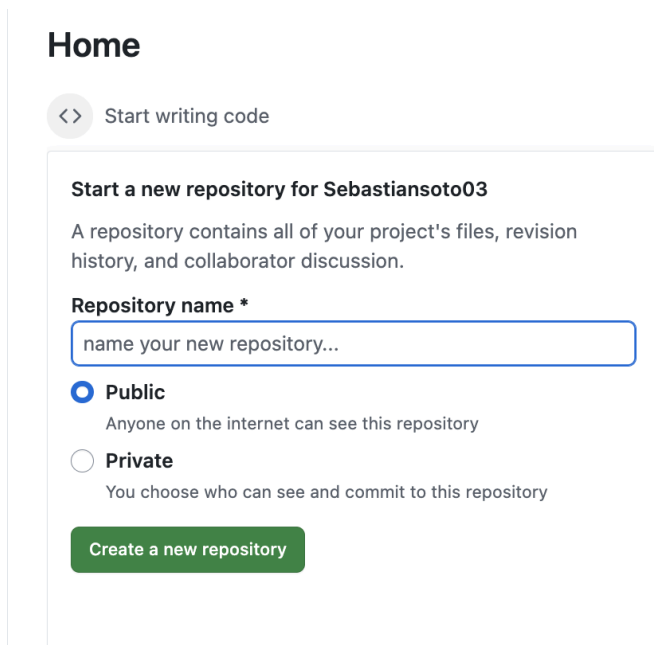
Paso 1: Haz clic en el ícono "+" en la esquina superior derecha y selecciona "New repository".

Paso 2: Completa los detalles del repositorio:

Nombre del repositorio: proyecto-ejemplo.

Descripción: Breve descripción del proyecto.

Visibilidad: Selecciona Público.



The screenshot shows the GitHub 'Start a new repository' interface. At the top, there's a 'Home' header and a button with a code icon and the text 'Start writing code'. Below this is a box titled 'Start a new repository for Sebastiansoto03'. Inside the box, there's a description: 'A repository contains all of your project's files, revision history, and collaborator discussion.' followed by the label 'Repository name *' and a text input field containing 'name your new repository...'. Below the input field are two radio button options: 'Public' (selected) with the subtext 'Anyone on the internet can see this repository', and 'Private' with the subtext 'You choose who can see and commit to this repository'. At the bottom of the box is a green button labeled 'Create a new repository'.

Paso 3: Buenas prácticas:

Inicializar con un README: Marca esta opción para proporcionar información básica del proyecto.

Agregar un .gitignore: Selecciona el lenguaje de programación que usarás para excluir archivos innecesarios.

Elegir una licencia: Selecciona una licencia adecuada para tu proyecto.

Paso 4: Haz clic en "Create repository".

2.2 Entendiendo la Estructura del Repositorio

Al crear el repositorio, serás redirigido a su página principal.

Observa las secciones clave:

Code: Contiene los archivos y directorios del proyecto.

Issues: Para reportar y rastrear errores o solicitudes de funciones.

Pull requests: Donde se gestionan las contribuciones externas.

Actions: Para automatizar flujos de trabajo (CI/CD).

Wiki y Projects: Para documentación y gestión de proyectos.

Parte 3: Gestión de Ramas - main, qa, y dev

3.1 Comprendiendo las Ramas

main: Rama principal y estable del proyecto.

qa (Quality Assurance): Rama para pruebas y verificación de calidad.

dev (Development): Rama donde se desarrollan nuevas funciones y cambios.

3.2 Creación de Ramas en GitHub

Paso 1: En la página del repositorio, haz clic en el menú desplegable de ramas que dice "main".

Paso 2: Escribe "qa" y presiona "Enter" para crear la nueva rama.

Paso 3: Repite el proceso para crear la rama "dev".

3.3 Buenas Prácticas en Gestión de Ramas

Separación de Entornos: Mantener ramas separadas para desarrollo, pruebas y producción.

Control de Calidad: Mergear (fusionar) cambios de dev a qa después de completar una función.

Revisiones de Código: Antes de fusionar a main, realizar revisiones de código y pruebas exhaustivas.

Parte 4: Clonación de un Repositorio

4.1 Clonar el Repositorio en tu Computadora

Paso 1: Abre la terminal o línea de comandos en tu computadora.

Paso 2: Navega hasta el directorio donde deseas almacenar el repositorio local:

bash

Copiar código

cd /ruta/a/tu/directorio

Paso 3: Obtén la URL del repositorio:

En GitHub, haz clic en el botón "Code" y copia la URL HTTPS o SSH.

Paso 4: Clona el repositorio:

Usando HTTPS:

bash

Copiar código

git clone https://github.com/tu_usuario/proyecto-ejemplo.git

Usando SSH (si configuraste claves SSH):

Paso 5: Verifica que el repositorio se clonó correctamente:

bash

Copiar código

cd proyecto-ejemplo

ls -la

4.2 Comprender el Proceso de Clonación

git clone: Este comando crea una copia local exacta del repositorio remoto.

Origen Remoto: El repositorio de GitHub es el "origen" remoto predeterminado.

Archivos Descargados: Todos los archivos y ramas del repositorio se descargan a

tu máquina local.

Parte 5: Configuración Local de Variables en Git

5.1 Configurar Usuario y Correo Electrónico Localmente

Para evitar afectar otras configuraciones globales, establecemos las variables de configuración localmente dentro del repositorio.

Paso 1: Dentro del directorio del repositorio, ejecuta:

bash

Copiar código

```
git config --local user.name "Tu Nombre"
```

```
git config --local user.email "tu_email@example.com"
```

Esto actualizará el archivo `.git/config` con tus datos para este repositorio específico.

5.2 Verificar Configuraciones Locales

Paso 2: Para verificar las configuraciones locales:

bash

Copiar código

```
git config --local --list
```

```
user.name=Tu Nombre
```

```
user.email=tu_email@example.com
```

5.3 Buenas Prácticas en Configuraciones Locales

Aislamiento de Configuraciones: Al configurar variables localmente, evitas conflictos con otros proyectos.

Privacidad: Puedes usar diferentes correos electrónicos o nombres de usuario en distintos repositorios.

Control: Las configuraciones locales tienen prioridad sobre las globales para ese repositorio.

Parte 6: Colaboración en Proyectos

6.1 Trabajar en la Rama dev

Paso 1: Cambia a la rama "dev":

bash

Copiar código

git checkout dev

Paso 2: Verifica que estás en la rama correcta:

bash

Copiar código

git branch

La rama actual estará marcada con un asterisco * dev.

Paso 3: Crea o modifica archivos en esta rama.

Paso 4: Agrega los cambios al área de preparación:

bash

Copiar código

git add nombre_del_archivo

Paso 5: Realiza un commit con un mensaje descriptivo:

bash

Copiar código

git commit -m "Agregar nueva función X en dev"

Paso 6: Sube los cambios a la rama "dev" en GitHub:

bash

Copiar código

git push origin dev

6.2 Fusionar Cambios de dev a qa

Paso 1: Cambia a la rama "qa":

bash

Copiar código

git checkout qa → git switch qa

Paso 2: Fusiona los cambios de "dev" en "qa":

bash

Copiar código

git merge dev

Si hay conflictos, Git te notificará y deberás resolverlos manualmente.

Paso 3: Sube los cambios a GitHub:

bash

Copiar código

git push origin qa

6.3 Fusionar Cambios de qa a main

Paso 1: Después de probar y verificar en "qa", cambia a la rama "main":

bash

Copiar código

git checkout main → git switch main

Paso 2: Fusiona los cambios de "qa" en "main":

bash

Copiar código

git merge qa

Paso 3: Sube los cambios a GitHub:

bash

Copiar código

git push origin main

6.4 Buenas Prácticas en Fusiones y Pull Requests

Revisiones de Código: Antes de fusionar a "main", utiliza pull requests para revisar los cambios.

Pull Requests Internos:

En GitHub, ve a la pestaña "Pull requests" y haz clic en "New pull request".

Selecciona "base: main" y "compare: qa".

Revisa los cambios y crea el pull request.

Asigna revisores si es necesario.

Una vez aprobado, puedes fusionar los cambios.

6.5 Realizar un Fork y Colaborar en Otro Proyecto

Paso 1: Ve al repositorio público que deseas contribuir, por ejemplo, github.com/ejemplo/proyecto-publico.

Paso 2: Haz clic en "Fork" en la esquina superior derecha.

Esto crea una copia del repositorio en tu cuenta.

Paso 3: Clona tu fork en tu máquina local:

```
bash
```

Copiar código

```
git clone https://github.com/Shoy777/Control-Atencion-Veterinaria.git
```

Paso 4: Configura el remoto upstream para mantener tu fork actualizado con el original:

```
bash
```

Copiar código

```
cd proyecto-publico
```

```
git remote add upstream https://github.com/ejemplo/proyecto-publico.git
```

Paso 5: Crea una rama para tus cambios:

```
bash
```

Copiar código

```
git checkout -b mi-contribucion → git switch
```

Paso 6: Realiza los cambios, agrega y haz commit:

bash

Copiar código

git add .

git commit -m "Corregir error en la función Y"

Paso 7: Sube tu rama a GitHub:

bash

Copiar código

git push origin mi-contribucion

Paso 8: Crea un pull request desde tu rama "mi-contribucion" hacia el repositorio original.

Ve a tu repositorio fork en GitHub.

Haz clic en "Compare & pull request".

Asegúrate de que el "base repository" es el original y el "head repository" es tu fork.

Proporciona un título y descripción claros.

Envía el pull request y espera comentarios o aprobación.

Conclusión

Este ejercicio detallado te ha guiado a través de:

Configuración Local de Git: Asegurando que las configuraciones no afecten otros proyectos.

Creación y Gestión de Repositorios con Buenas Prácticas: Incluyendo la inicialización

adecuada y uso de archivos esenciales.

Gestión de Ramas: Trabajando con main, qa, y dev para separar entornos y mantener la

estabilidad.

Clonación Detallada de Repositorios: Comprendiendo cada paso y su importancia.

Colaboración en Proyectos: Utilizando fork, clone, y pull request para contribuir efectivamente.

Notas Finales y Buenas Prácticas Adicionales:

Mensajes de Commit Claros:

Sigue la convención de usar el presente imperativo: "Agrega", "Corrige", "Elimina".

Proporciona contexto si es necesario.

Resolución de Conflictos:

Lee cuidadosamente los conflictos que Git resalta.

Usa herramientas como git mergetool o editores con soporte para Git.

Sincronización Regular:

Mantén tu repositorio local y fork sincronizados con el original para evitar conflictos grandes.

Documentación:

Actualiza el README y otros documentos para reflejar cambios significativos.

Añade comentarios en el código donde sea necesario.

Seguridad:

No cometas información sensible como contraseñas o claves.

Usa archivos de configuración de ejemplo y excluye los reales con `.gitignore`.

¡Felicidades por completar este ejercicio mejorado! La práctica constante y la adherencia a buenas prácticas te convertirán en un colaborador eficaz y valorado en cualquier proyecto.