Centro de Desarrollo

Actividad #2. Haz tu propio repositorio en GitHub

En esta actividad de permanencia y capacitación en el Centro de Desarrollo, aprenderás a utilizar una de las tecnologías más importantes dentro del mundo del desarrollo de software: Git.

Git es una tecnología que nos permite llevar un control de versiones en los archivos, documentos y código fuente que tengamos dentro de cualquier proyecto. En esencia, imagínate que es algo similar a Google Drive, OneDrive o Dropbox, pero especializado para el desarrollo de software.

En esta actividad podrás:

* Aprender los conceptos básicos de Git
* Tener tu propia cuenta Pro en GitHub
* Hacer tus primeros commit y push hacia un repositorio
* Crear el espacio en el que irás construyendo tu proyecto Así pues, para esta actividad, deberás realizar lo siguiente:
  1. Ve el siguiente video para que comiences a entrar en contexto sobre lo que es **git** y para qué nos sirve: https://[www.youtube.com/watch?v=jGehuhFhtnE](http://www.youtube.com/watch?v=jGehuhFhtnE)
  2. Una vez que hayas terminado, ve a la página oficial de **git** y descarga e instala la versión que corresponda a tu sistema operativo: https://git-scm.com/ durante la instalación, deja las opciones que se encuentran por defecto.
  3. Dirígete a **GitHub** (https://github.com/) y crea una cuenta **utilizando tu correo institucional** (@alumnos.uaq.mx). Si no cuentas con correo institucional, haz la cuenta con un correo normal, pero **no podrás realizar el siguiente paso**.
  4. Ahora, mejorarás tu cuenta en GitHub, aprovechando tu cuenta de correo institucional. Ingresa a https://education.github.com/pack y sigue los pasos que te pide para que tu cuenta ahora sea Pro.

Hasta este punto, ya deberías de tener todo lo indispensable para realizar la parte práctica de esta segunda actividad.

Actividades para desarrollar:

1. **Crear un repositorio local**: Crea un nuevo directorio en tu computadora y conviértelo en un repositorio Git utilizando el comando **git init**. Agrega algunos archivos al repositorio y realiza un commit con los cambios.
2. **Clonar un repositorio**: Clona un repositorio existente desde GitHub a tu computadora utilizando el comando **git clone <URL\_del\_repositorio>**. Asegúrate de estar autenticado con tu cuenta de GitHub antes de hacer esto.
3. **Branching**: Crea una nueva rama (branch) en tu repositorio local utilizando el comando **git branch <nombre\_rama>**. Cambia a esa rama utilizando **git checkout <nombre\_rama>**. Realiza algunos cambios y commits en esa rama.
4. **Merging**: Luego de realizar cambios en una rama, regresa a la rama principal (por ejemplo, **main** o **master**) utilizando **git checkout main** y fusiona los cambios de la rama secundaria utilizando **git merge <nombre\_rama>**. Resuelve cualquier conflicto si es necesario.
5. **Remotos y Push**: Crea un repositorio en GitHub e inicia un repositorio remoto en tu repositorio local utilizando el comando **git remote add origin**

**<URL\_del\_repositorio\_remoto>**. Luego, empuja (push) tus cambios locales al repositorio remoto con **git push origin main** (o el nombre de tu rama principal).

1. **Pull Request**: Haz un cambio en tu repositorio remoto directamente desde la interfaz web de GitHub. Luego, desde tu repositorio local, asegúrate de

que estás en la rama principal y ejecuta **git pull origin main** para sincronizar tus cambios locales con el repositorio remoto. A continuación, crea un Pull Request en GitHub para fusionar tus cambios en la rama principal.

1. **Revisar Pull Request**: Invita a otro colaborador (puede ser otro compañero en la actividad) a tu repositorio y pídele que haga un cambio en una rama secundaria y envíe un Pull Request. Revisa su Pull Request, comenta sobre los cambios y, si es apropiado, aprueba y fusiona los cambios en la rama principal.
2. **Revertir cambios**: Haz cambios en tu repositorio local y realiza un commit. Luego, utiliza el comando **git revert <ID\_del\_commit>** para revertir los cambios realizados en ese commit en particular.
3. **Rebase**: Experimenta con el comando **git rebase** para reorganizar y combinar commits antes de hacer un Pull Request.
4. **Gestión de conflictos**: Crea deliberadamente conflictos en los archivos al hacer cambios en diferentes ramas y luego intenta resolverlos utilizando las herramientas proporcionadas por Git.

Actividades adicionales que realizar:

1. **Visualizar historial de commits**: Utiliza el comando **git log** para visualizar el historial de commits en tu repositorio. Prueba diferentes opciones como **git log --oneline**, **git log --author**, o **git log --since** para filtrar y ver información específica.
2. **Deshacer cambios sin perderlos**: Utiliza el comando **git stash** para guardar temporalmente los cambios que no deseas commitear en el momento. Luego, puedes aplicarlos de nuevo con **git stash apply** o **git stash pop**.
3. **Ignorar archivos con .gitignore**: Crea un archivo **.gitignore** en tu repositorio para indicar a Git qué archivos o directorios no deben ser

rastreados. Agrega algunos patrones de archivos que no deseas incluir en tu repositorio.

1. **Etiquetas (tags)**: Crea una etiqueta en un commit específico utilizando el comando **git tag <nombre\_etiqueta>**. Luego, explora cómo listar y ver las etiquetas en tu repositorio.
2. **Reescribir historial**: Utiliza comandos como **git commit --amend** para editar el mensaje de un commit, o **git rebase -i HEAD~n** para combinar, reorganizar o eliminar commits en el historial.
3. **Ver diferencias entre commits**: Utiliza **git diff** para ver las diferencias entre commits o ramas. Prueba diferentes opciones como **git diff**

**<commit1> <commit2>** o **git diff <rama1> <rama2>**.

1. **Borrar ramas locales**: Aprende a borrar ramas locales que ya no necesitas con el comando **git branch -d <nombre\_rama>**.
2. **Borrar ramas remotas**: Aprende a borrar ramas remotas que ya no son necesarias con el comando **git push --delete <nombre\_rama>**.
3. **Actualizar repositorio local**: Utiliza **git pull** para actualizar tu repositorio local con los cambios realizados en el repositorio remoto.
4. **Clonar un repositorio específico**: Aprende a clonar solo una rama específica de un repositorio remoto utilizando **git clone --single-branch -- branch <nombre\_rama> <URL\_del\_repositorio>**.
5. **Restaurar versiones anteriores**: Utiliza **git checkout**

**<nombre\_commit>** para volver a versiones anteriores de tu código.

1. **Ver cambios antes del merge**: Utiliza **git diff**

**<rama\_destino>..<rama\_fuente>** para visualizar los cambios que se fusionarán antes de realizar el merge.

1. **Ignorar cambios locales**: Aprende a descartar los cambios locales y volver al último commit utilizando **git checkout -- .**.
2. **Guardar cambios parcialmente**: Usa **git add -p** para seleccionar y guardar cambios específicos dentro de un archivo.
3. **Comandos útiles**: Investiga otros comandos útiles como **git cherry-pick**, **git bisect**, **git reflog**, y **git submodule** para ampliar tu conocimiento y comprensión de Git.

Recuerda que estas actividades te ayudarán a familiarizarte con los comandos básicos de Git y a desarrollar tus habilidades para colaborar en proyectos a través de GitHub. Practicar estas tareas te permitirá mejorar tus habilidades en el desarrollo de software y trabajar de manera colaborativa en equipos de desarrollo. ¡Diviértete explorando y aprendiendo!