Presentación

Nombre: Anderly Nicole

Apellido: Villanueva Belén

Matricula: 2020-9506

Profesor: Kelyn Tejada

Materia: Programación 3

Instituto: Instituto tecnológico de las américa (ITLA)

Tema: Tarea 3

1- Desarrolla el siguiente Cuestionario

1-Que es Git?

Es un software de control de versiones, El control de versiones distribuido permite a los desarrolladores descargar un software, realizar cambios y subir la versión que han modificado.

2-Para que funciona el comando Git init?

Es un comando que crea un subdirectorio nuevo llamado. git, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git.

4-Que es una rama?

Las ramas de Git son un puntero eficaz para las instantáneas de tus cambios. Cuando quieres añadir una nueva función o solucionar un error, independientemente de su tamaño, generas una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que el código inestable se fusione con el código base principal, y te da la oportunidad de limpiar tu historial futuro antes de fusionarlo con la rama principal.

3-Como saber es que rama estoy?

Con el comando git status

5-Quien creo git?

Linus Torvalds

6-Cuales son los comandos más esenciales de Git?

- **git init:**

Esto crea un subdirectorio nuevo llamado. git, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento**.**

- **git fetch:**

Descarga los cambios realizados en el repositorio remoto**.**

git merge <nombre_rama>:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama “nombre_rama”.

git pull:

Unifica los comandos fetch y merge en un único comando**.**

git commit -m "<mensaje>":

Confirma los cambios realizados. El “mensaje” generalmente se usa para asociar al commit una breve descripción de los cambios realizados.

git push origin <nombre_rama>:

Sube la rama “nombre_rama” al servidor remoto**.**

git status:

Muestra el estado actual de la rama, como los cambios que hay sin commitear**.**

git add <nombre_archivo>:

Comienza a trackear el archivo “nombre_archivo”.

git checkout -b <nombre_rama_nueva>:

Crea una rama a partir de la que te encuentres parado con el nombre “nombre_rama_nueva”, y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

git checkout -t origin/<nombre_rama>:

Si existe una rama remota de nombre “nombre_rama”, al ejecutar este comando se crea una rama local con el nombre “nombre_rama” para hacer un seguimiento de la rama remota con el mismo nombre.

git branch:

Lista todas las ramas locales.

git branch -a:

Lista todas las ramas locales y remotas.

git branch -d <nombre_rama>:

Elimina la rama local con el nombre “nombre_rama”.

git push origin <nombre_rama>:

Commitea los cambios desde el branch local origin al branch “nombre_rama”.

git remote prune origin:

Actualiza tu repositorio remoto en caso de que algún otro desarrollador haya eliminado alguna rama remota.

git reset --hard HEAD:

Elimina los cambios realizados que aún no se hayan hecho commit.

git revert <hash_commit>:

Revierte el commit realizado, identificado por el “hash_commit”.

7-Que es git Flow?

Gitflow es un modelo alternativo de creación de ramas en Git en el que se utilizan ramas de función y varias ramas principales. Fue Vincent Driessen envien quien lo publicó por primera vez y quien lo popularizó.

Gitflow puede utilizarse en proyectos que tienen un ciclo de publicación programado, así como para la práctica recomendada de DevOps de entrega continua.

8- Que es trunk-based development?

es una estrategia de Git donde existe un trunk(un branch principal, usualmente llamado master/main) en el cual todo el equipo colabora he integra directamente (hace push), siguiendo estas consideraciones:

1- No existen branches de larga duración. No se le da mantenimiento a ningún branch. Por ejemplo en el caso que mencione anteriormente sobre la creación del branch de release (si fuese necesario crearlo) y se encuentra un error, el mismo debe ser reparado en el trunk y luego se haría un nuevo branch de release, pero nunca se le da mantenimiento a ese branch.
2- Se debe hacer commit al menos una vez al día (esto no significa que vamos integrar cualquier código solo por hacer commit, el siguiente punto lo explica mejor).
3- Todo lo que se le haga commit es código funcional, esto implica que se cumpla la definición de hecho (definition of done), se hayan creado las pruebas necesarias y todo lo que sea requerido para asegurarse que el código no esta introduciendo un bug 🐛.
4- El trunk siempre debe encontrarse en un estado verde y optimo con esto quiero decir listo para hacerle release.

Entonces, ¿En trunk based development no se hace revisión de código o pull request?

Claroooo que sí, pero principalmente esta metodología anima que haya más pair programming y no tener que hacer pull requests.

¿Debo utilizarlo?

Sí, pero tienes que saber:

Si la frecuencia con la que haces release es alta, trunk based development es la estrategia adecuada porque el proceso se convierte nada mas hacer el release.

Equipos y productos que ya se encuentran bien establecidos, con sus tiempos y fechas de release bien definidos es probable que no lo necesiten, la simplicidad de GitFlow es suficiente.

2- Desarrolle un ejercicio práctico en Azure Devops o GitHub con las siguientes características

Github: https://github.com/anderly01/popular-challenge