Presentación

Nombre

Lisbeth De jesus G.

Matrícula

2023-0287

Materia

Programación III

Asignación

Tarea 3

Docente

Kelyn Tejada Belliard

Sección

#9

Tabla de contenido

Introducción

En el mundo del desarrollo de software, el control de versiones es una herramienta fundamental que permite a los programadores gestionar y rastrear los cambios en su código de manera eficiente. Git, un sistema de control de versiones distribuido creado por Linus Torvalds en 2005, se ha convertido en el estándar de facto para la colaboración en proyectos de programación.

Este cuestionario explora los conceptos básicos de Git, incluyendo su funcionamiento, comandos esenciales, ramas, y metodologías de desarrollo como Git Flow y Trunk Based Development.

A través de estas preguntas y respuestas, se busca proporcionar una comprensión sólida de cómo utilizar Git para optimizar el flujo de trabajo y mejorar la productividad en el desarrollo de software.

Desarrolla el siguiente Cuestionario

1- ¿Qué es Git?

Git es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. Los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan la copia del repositorio con la del servidor. Este paradigma es distinto del control de versiones centralizado, donde los clientes deben sincronizar el código con un servidor antes de crear nuevas versiones.

2- ¿Para que funciona el comando Git init?

El comando git init crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío. La mayoría de los demás comandos de Git no se encuentran disponibles fuera de un repositorio inicializado, por lo que este suele ser el primer comando que se ejecuta en un proyecto nuevo.

Al ejecutar git init, se crea un subdirectorio de .git en el directorio de trabajo actual, que contiene todos los metadatos de Git necesarios para el nuevo repositorio. Estos metadatos incluyen subdirectorios de objetos, referencias y archivos de plantilla. También se genera un archivo HEAD que apunta a la confirmación actualmente extraída.

Aparte del directorio de .git, en el directorio raíz del proyecto, se conserva un proyecto existente sin modificar (a diferencia de SVN, Git no requiere un subdirectorio de .git en cada subdirectorio).

De manera predeterminada, git init inicializará la configuración de Git en la ruta del subdirectorio de .git. Puedes cambiar y personalizar dicha ruta si quieres que se encuentre en otro sitio. Asimismo, puedes establecer la variable de entorno $GIT_DIR en una ruta personalizada para que git init inicialice ahí los archivos de configuración de Git. También puedes utilizar el argumento --separate-git-dir para conseguir el mismo resultado. Lo que se suele hacer con un subdirectorio de .git independiente es mantener los archivos ocultos de la configuración del sistema (.bashrc, .vimrc, etc.) en el directorio principal, mientras que la carpeta de .git se conserva en otro lugar.

3- ¿Que es una rama?

Las ramas son una de las principales utilidades que disponemos en Git para llevar un mejor control del código. Se trata de una bifurcación del estado del código que crea un nuevo camino de cara a la evolución del código, en paralelo a otras ramas que se puedan generar

Las ramas nos pueden servir para muchos casos de uso. Por ejemplo, tras la elección del hosting para crear una página web o desarrollar una tienda online, para la creación de una funcionalidad que queramos integrar en un programa y para la cual no queremos que la rama principal se vea

afectada. Esta función experimental se puede realizar en una rama independiente, de modo que, aunque tardemos varios días o semanas en terminarla, no afecte a la producción del código que tenemos en la rama principal y que permanecerá estable.

El trabajo con ramas resulta muy cómodo en el desarrollo de proyectos, porque es posible que todas las ramas creadas evolucionen al mismo tiempo, pudiendo el desarrollador pasar de una rama a otra en cualquier momento según las necesidades del proyecto. Si en un momento dado el trabajo con una rama nos ha resultado interesante, útil y se encuentra estable, entonces podemos fusionar ramas para incorporar las modificaciones del proyecto en su rama principal.

En todo proyecto creado con Git podemos ver la rama en la que estamos actualmente situados.

4- ¿Cómo saber en cual rama estoy?

Una vez que se crea un repositorio en Git, se inicia automáticamente con el branch principal, también conocido como el branch master. Además, es posible crear branch a partir del master, que se denominan branches secundarios.

Para verificar todas las ramas del repositorio, se utiliza el comando Git Branch. El comando Git para crear un branch es el siguiente: Git branch <nombre del branch>. Aquí, es importante recordar que la rama se crea de acuerdo al branch en el que te encuentras en el momento que escribes un comando, así que ten en cuenta este hecho.

Vale la pena señalar que, para que Git sepa dónde están los branches, la herramienta utiliza asteriscos (*). Si deseas ir a una rama específica, simplemente utiliza git checkout <nombre del branch>. Finalmente, es posible eliminar un branch utilizando el comando <git branch -d <nombre del branch>.

5-¿Quien creo git?

Fue creado por Linus Torvalds, el creador del kernel de Linux, en 2005. Git se caracteriza por su eficiencia, confiabilidad y compatibilidad con grandes proyectos de código fuente.

Linus Torvalds es un programador finlandés que es considerado uno de los padres fundadores del software libre. En 1991, Torvalds comenzó a desarrollar el kernel de Linux, el núcleo del sistema operativo GNU/Linux. Para poder gestionar las versiones del código fuente de Linux, Torvalds utilizó el sistema de control de versiones BitKeeper. Sin embargo, en 2005, los desarrolladores de BitKeeper decidieron restringir el uso de su software para proyectos comerciales.

Esto llevó a Torvalds a crear su propio sistema de control de versiones, que llamó Git. Git se basó en los conceptos de control de versiones distribuidos, que permitían a los desarrolladores trabajar en el mismo proyecto desde diferentes ubicaciones.

6-¿Cuáles son los comandos más esenciales de Git?

Git status: Muestra el estado del directorio en el que estás trabajando y la instantánea preparada.

Git revert: Permite deshacer una instantánea confirmada.

Git reset: Deshace los cambios en los archivos del directorio de trabajo.

Git remote: Es un comando útil para administrar conexiones remotas.

Git reflog: Git realiza el seguimiento de las actualizaciones en el extremo de las ramas mediante un mecanismo llamado registro de referencia o reflog. Esto permite volver a los conjuntos de cambios aunque no se haga referencia a ellos en ninguna rama o etiqueta.

Git rebase -i: La marca -i se usa para iniciar una sesión de cambio de base interactivo. Esto ofrece todas las ventajas de un cambio de base normal, pero te da la oportunidad de añadir, editar o eliminar confirmaciones sobre la marcha.

Git rebase: Un cambio de base con git rebase permite mover las ramas, lo que ayuda a evitar confirmaciones de fusión innecesarias.

Git push: Enviar (push) es lo opuesto a recuperar (fetch), con algunas salvedades. Permite mover una o varias ramas a otro repositorio, lo que es una buena forma de publicar contribuciones. Es como svn commit, pero envía una serie de confirmaciones en lugar de un solo conjunto de cambios.

Git pull: Este comando es la versión automatizada de git fetch. Descarga una rama de un repositorio remoto e inmediatamente la fusiona en la rama actual. Este es el equivalente en Git de svn update.

Git merge: Es una forma eficaz de integrar los cambios de ramas divergentes. Después de bifurcar el historial del proyecto con git branch, git merge permite unirlo de nuevo.

Git log: Permite explorar las revisiones anteriores de un proyecto. Proporciona varias opciones de formato para mostrar las instantáneas confirmadas.

Git init: Inicializa un nuevo repositorio de Git. Si quieres poner un proyecto bajo un control de revisiones, este es el primer comando que debes aprender.

Git fetch: Con este comando, se descarga una rama de otro repositorio junto con todas sus confirmaciones y archivos asociados. Sin embargo, no intenta integrar nada en el repositorio local. Esto te permite inspeccionar los cambios antes de fusionarlos en tu proyecto.

Git config: Este comando va bien para establecer las opciones de configuración para instalar Git. Normalmente, solo es necesario usarlo inmediatamente después de instalar Git en un nuevo equipo de desarrollo.

Git commit –amend: Pasar la marca --amend a git commit permite modificar la confirmación más reciente.

Git commit: Confirma la instantánea preparada en el historial del proyecto. En combinación con git add, define el flujo de trabajo básico de todos los usuarios de Git.

Git clone: Crea una copia de un repositorio de Git existente.

Git clean: Elimina los archivos sin seguimiento de tu directorio de trabajo. Es la contraparte lógica de git reset, que normalmente solo funciona en archivos con seguimiento.

Git Checkout: Además de extraer las confirmaciones y las revisiones de archivos antiguas, git checkout también sirve para navegar por las ramas existentes.

Git add: Mueve los cambios del directorio de trabajo al área del entorno de ensayo.

7- ¿Que es git Flow?

Es un flujo de trabajo basado en Git que brinda un mayor control y organización en el proceso de integración continua.

Git Flow Aumenta la velocidad de entrega de código terminado al equipo de pruebas, Disminuyen los errores humanos en la mezcla de las ramas, Elimina la dependencia de funcionalidades al momento de entregar código para ser puesto en producción, El equipo de trabajo está conformado por más de dos (2) personas, Se emplean metodologías ágiles, El proyecto tiene cambios frecuentes y se requiere actualizar el ambiente de producción garantizando continuidad en la operación, El proyecto tiene un nivel de complejidad considerable, Se desea tener un proceso de soporte a errores efectivo con actualizaciones rápidas.

Deben existir dos ramas principales para que el flujo de trabajo funcione correctamente:

master

develop

GitFlow creará por defecto los siguiente prefijos para las ramas auxiliares, los cuales ayudan a identificar y tener control en el repositorio:

feature/

release/

hotfix/

bugfix/

support/

Se recomienda agregar un prefijo a las etiquetas, por ejemplo, la letra «***v***» sin comillas.

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:

No existen branches de larga duración. No se le da mantenimiento a ningún branch. Nota: en algunos casos es permitido hacer cherry-pick para mover un cambio, o hacerlo directo en el branch de release, pero se debe de mover inmediatamente al branch de develop.

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). Esto lo que busca es eliminar la distancia entre los desarrolladores cuando se empieza a codear cosas nuevas.

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 . Esto significa que el equipo debe de tener un grado de madurez y responsabilidad alto al momento de entregar el código.

El trunk siempre debe encontrarse en un estado verde y optimo con esto quiero decir listo para hacerle release.

Conclusion

En conclusión, Git no solo es una herramienta poderosa para el control de versiones, sino que también promueve un entorno de colaboración eficiente entre desarrolladores.

Con su capacidad para gestionar ramas y facilitar la integración continua, Git se adapta a diversas metodologías de desarrollo, como Git Flow y Trunk Based Development, que optimizan el proceso de entrega de software.

Al dominar los comandos y conceptos fundamentales de Git, los programadores pueden trabajar de manera más organizada y efectiva, asegurando que sus proyectos se mantengan en un estado óptimo y libre de errores.

La comprensión y aplicación de estas prácticas son esenciales para cualquier desarrollador que busque sobresalir en el dinámico campo de la programación.

Bibliografía

arsys. (19 de abril de 2024). *arsys*. Obtenido de arsys: https://www.arsys.es/blog/ramas-git

atlassian. (11 de enero de 2021). *atlassian*. Obtenido de atlassian: https://www.atlassian.com/es/git/glossary#commands

atlassian. (02 de abril de 2022). *atlassian*. Obtenido de atlassian: https://www.atlassian.com/es/git/tutorials/setting-up-a-repository/git-init

gfourmis. (10 de Noviembre de 2023). *gfourmis*. Obtenido de gfourmis: https://gfourmis.co/gitflow-sin-morir-en-el-intento/

HostGator. (16 de agosto de 2023). *HostGator*. Obtenido de HostGator: https://www.hostgator.mx/blog/git-branch-que-es/

leninmhs. (12 de Noviembre de 2023). *leninmhs*. Obtenido de leninmhs: https://leninmhs.com/historia-de-git/

Microsoft. (04 de Octubre de 2023). *Microsoft*. Obtenido de Microsoft: https://learn.microsoft.com/es-es/devops/develop/git/what-is-git#built-in-integration

CR, M. Á.. (13 de Octubre de 2020). *dev*. Obtenido de dev: https://dev.to/marianocodes/porque-trunk-based-development-i5n