Desarrollo Web Entorno Desarrollo

Fase de Implementación

Revisión de código

La revisión de código es una de las fases más importantes de todo desarrollador, ya que una vez finalizada una primera versión local de nuestro desarrollo, es necesario revisar el código para identificar posibles problemas o soluciones a otros problemas. Para ello existen diferentes métodos para depurar el código.

Método de depuración del patito de goma

El método de depuración del patito de goma es un término informal utilizado en ingeniería de software para describir un método de revisión de código. El nombre hace referencia a una historia del libro “The Pragmatic Programmer” en donde un programador toma un patito de goma y revisa su código forzándose a sí mismo a explicarlo, línea por línea, al pato. Existen otros muchos términos para esta técnica, que a menudo tienen que ver con objetos inanimados.

Muchos programadores han tenido la experiencia de explicar un problema de programación a alguien más, posiblemente a alguien que no sabe nada sobre programación, y encontrar la solución en el proceso de explicar el problema. Al comparar lo que supuestamente hace el código con lo que hace en realidad, cualquier incongruencia resulta evidente. Usando un objeto inanimado, el programador puede tratar de lograr el mismo efecto sin tener que hablar con otra persona.

Herramienta de control de versiones: Git

Es un software de control de versiones diseñado pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de PC incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

El diseño de Git mantiene una enorme cantidad de código distribuida y gestionada por mucha gente, que incide en numerosos detalles de rendimiento, y de la necesidad de rapidez en una primera implementación.

Entre las características más relevantes se encuentran:

- Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal.
- Gestión distribuida. Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.
- Almacenamiento de la información menos pesada debido al guardado de cambios y no de archivos completos.
- Los almacenes de información pueden publicarse.
- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.
- Compatibilidad con una amplia gama de herramientas de desarrollo.

Existen muchas órdenes de git que nos sirven para realizar diferentes funcionalidades:

- 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. Comprueba el estado del repositorio local en comparación con el repositorio remoto.
- git merge: 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: Confirma los cambios realizados. El “mensaje” generalmente se usa para asociar al commit una breve descripción de los cambios realizados.
- git push: Sube la rama “nombre_rama” al servidor remoto.
- git status: Muestra el estado actual de la rama, como los cambios que hay sin commitear.

Entre muchas otras órdenes que no son necesarias para este apartado teórico. Se adjuntan plantillas resumen de estas funcionalidades.

Metodología de control de versiones: Git Flow

Cuando evalúas un flujo de trabajo para tu equipo, lo más importante es que tengas en cuenta la cultura de tu equipo. Quieres que el flujo de trabajo mejore la eficacia de tu equipo, no que sea una carga que limite la productividad.

Estas son algunas cosas que se deben tener en cuenta a la hora de evaluar un flujo de trabajo de Git:

- ¿Este flujo de trabajo se escala acorde al tamaño del equipo?
- ¿Es fácil deshacer los errores y los fallos con este flujo de trabajo?
- ¿Este flujo de trabajo impone al equipo excesos cognitivos nuevos e innecesarios?

Git flow en concreto es el flujo de trabajo más conocido. Está pensado para aquellos proyectos que tienen entregables y ciclos de desarrollo bien definidos. Está basado en dos grandes ramas con infinito tiempo de vida (ramas master y develop) y varias ramas de apoyo, unas orientadas al desarrollo de nuevas funcionalidades (ramas feature-*), otras al arreglo de errores (hotfix-*) y otras orientadas a la preparación de nuevas versiones de producción (ramas release-*). La metodología gitflow facilita la automatización de las tareas implicadas en este flujo de trabajo.

A continuación se describen cada una de las ramas que trata Git flow:

Master

Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.

Development

Es una rama sacada de Master. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego de que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir, que la rama se encuentre estable, se puede hacer un merge de development sobre la rama Master.

Features

Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de Development. Una vez que la funcionalidad esté desarrollada, se hace un merge de la rama sobre Development, donde se integrará con las demás funcionalidades.

Hotfix

Son errores de software que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de Master. Una vez corregido el error, se debe hacer una unificación de la rama sobre Master. Al final, para que no quede desactualizada, se debe realizar la unificación de Master sobre Development.

Release

Las ramas de release apoyan la preparación de nuevas versiones de producción. Para ellos se arreglan muchos errores menores y se preparan adecuadamente los metadatos. Se suelen originar de la rama develop y deben fusionarse en las ramas master y develop.