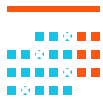
Blog

Tecnología para Desarrollo

Tecnología para Desarr...

Tecnología para Negocio



Eventos y seminarios

Primeros pasos con Git

por Federico Salomone (https://www.paradigmadigital.com/author/fsalomone/)

Madrid, España 5 de septiembre del 2016 git

sin comentarios

La gestión de la configuración es un área de la Ingeniería de Software cuya misión es **controlar la evolución de un producto desarrollado**. Involucra un conjunto de técnicas para gestionar con eficiencia los cambios que se realicen sobre ese producto a lo largo de su ciclo de vida.

El objetivo de este artículo es hacer una **breve introducción general al uso estandarizado de Git** en entornos colaborativos utilizando solicitudes de integración y Git como sistema controlador de versiones.

(https://www.paradigmadigital.com/wp-content/uploads/2016/08/Git1.jpg)

Uno de los aspectos que administra la gestión de la configuración es el **control de versiones**, que se puede definir como el proceso responsable de identificar y mantener registros de las diversas versiones y liberaciones de un sistema. En términos generales, cuando se habla de gestión de versiones en un proyecto de software, se refiere a la **administración del código fuente**.

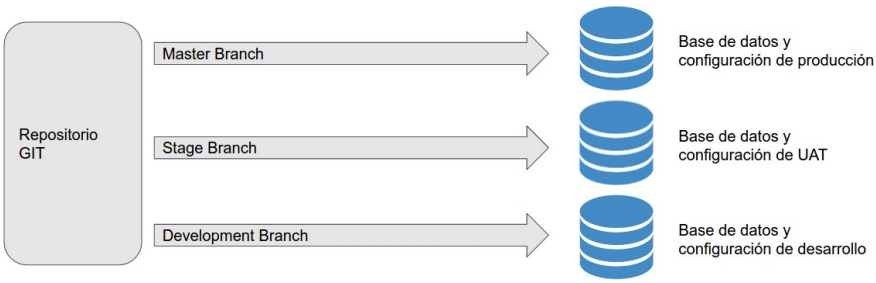
Por suerte, para el mundo del desarrollo del software existen herramientas que ayudan a los equipos de desarrollo a manejar los cambios en el código a lo largo del ciclo de vida. Estos sistemas mantienen el registro de cada modificación realizada en un repositorio o base de datos con el objetivo de poder recuperarlas fácilmente, tratando de interrumpir lo menos posible al resto de los miembros del equipo.

El sistema más utilizado mundialmente en la actualidad es Git (https://git-scm.com/), que fue desarrollado en el año 2005 por Linus Torvalds (https://es.wikipedia.org/wiki/Linus_Torvalds) (el creador del kernel de Linux) y es un proyecto open source.

Configuración de ramas del proyecto

Cuando comienza un proyecto resulta necesario realizar una configuración inicial de ramas o branches. Una rama representa una línea independiente de desarrollo que mantiene su propia historia de commits y revisiones.

Las **ramas recomendadas** a crear al empezar el desarrollo son:



(https://www.paradigmadigital.com/wp-content/uploads/2016/08/Git2.jpg)

Cada rama debería tener su propio entorno ya sea de base de datos como de configuración (propiedades de archivos properties, endpoints, datos de conexión a base de datos, etc).

- **Master branch**: Es la rama que contiene la última versión del producto estable. No es de uso diario y debería modificarse solamente al lanzar una nueva versión del producto.
- **Stage branch**: Es la rama comúnmente denominada UAT / Release. Es aquella en la cual los usuarios o el cliente realizan sus test de aceptación y validan aquellas funcionalidades que fueron desarrolladas o la resolución de algún bug.
- **Development branch**: Es la rama sobre la cual los desarrolladores van añadiendo las nuevas funcionalidades terminadas. Todos los desarrolladores deberían trabajar de forma frecuente con esta rama, accediendo a la última versión de desarrollo y volcando nuevas funcionalidades creadas.

También puede existir una rama denominada "**Hot fix**" con el objetivo de solucionar cualquier bug que haya surgido en la versión de producción y es necesario resolver inmediatamente.

Configuración de ramas de trabajo diario

Los desarrolladores que participen en el proyecto deberán crear ramas de trabajo "temporales" por cada funcionalidad o bug en el que trabajen.

Es muy común, en el caso de estar usando una herramienta de gestión de proyecto como pueden ser Jira (https://es.atlassian.com/software/jira) o Trello (https://trello.com/) que el nombre de la rama, se corresponda con el título de la historia de usuario, tarea o bug a resolver.

Por ejemplo, si tenemos una historia de usuario con el ID C-181 en la que se solicita: "*Como usuario quiero poder ver un listado de clientes para analizar sus características*", se puede crear una nueva rama denominada "**C-181-ListarClientes**". También resulta bastante común en el caso de no contar con identificadores o sistema de gestión, utilizar el prefijo f/ para funcionalidades y b/ para bugs. Por ejemplo: **f/ListarClientes o b/SumaIncorrectaIVA**.

Estas ramas de trabajo siempre deberán partir de la rama "Development" que hemos definido anteriormente y, una vez terminada la funcionalidad, deberán fusionarse nuevamente con ésta, agregando la nueva funcionalidad o resolución de bug. Para esta última actividad, el programador realizará lo que se llama una solicitud de integración o Pull Request.

Trabajando con Solicitudes de integración / Pull Requests

Una **solicitud de integración** es un mecanismo colaborativo a través del cual un desarrollador puede avisar al resto del equipo que ha completado una funcionalidad y que está apta de ser revisada por el equipo e integrada al repositorio de código. Esta acción se realiza solicitando la integración del branch utilizado para hacer la funcionalidad o corregir el bug hacia el branch sobre el cual trabaja el equipo que, según la arquitectura propuesta en este artículo, sería el de desarrollo.

Antes de realizar la solicitud, el programador debe verificar que la versión que propone subir no entra en conflicto con ningún elemento de la rama de Development. Es recomendable, entonces, realizar una **acción de rebase** (puede usarse también merge, pero con el objetivo de mantener un historial limpio y claro, se recomienda el uso de rebase) para integrar cualquier otra funcionalidad que se haya añadido durante el tiempo de desarrollo de la tarea.

Es importante entender que el **Pull Request** es mucho más que un aviso de finalización de una funcionalidad, es una oportunidad de inspección para el equipo en donde colectivamente se hace una revisión del código y se proponen mejoras en el caso que corresponda. Muchas veces los equipos usan las solicitudes de integración como mecanismo para demostrar pruebas de concepto a sus compañeros sin la necesidad de mezclar su código con el producto que se está desarrollando.

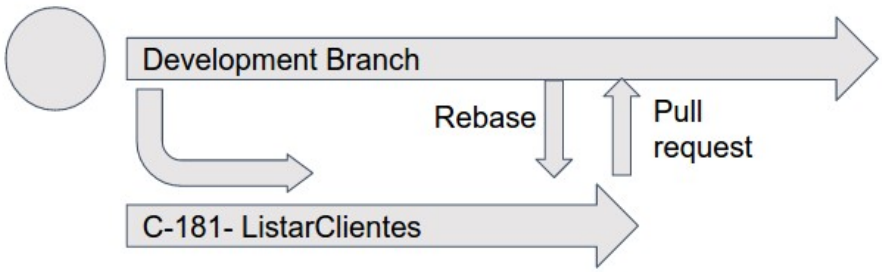
En tal caso, se pueden tratar temas como: una forma innovadora de realizar un proceso, demostrar el uso de una nueva librería, etc. Es condición necesaria que esta revisión del código se realice desde **la perspectiva del respeto** (fundamentalmente en equipos virtuales donde no se puede discutir la revisión cara a cara) para que ningún miembro del equipo se sienta ofendido por los comentarios.

En forma resumida, **los pasos deberían ser los siguientes**:

1. Crear una **rama propia** de la tarea o feature a completar desde Development respetando la nomenclatura definida para el proyecto.
2. Trabajar y completar la tarea.
3. **Comprobar que no exista nada nuevo en Development** a incorporar en la rama de trabajo a través de una operación de rebase.
4. **Solucionar los conflictos de código** surgidos en el punto anterior y actualizar los cambios en la rama de trabajo.
5. **Generar un Pull Request** a la rama de Development.
6. El equipo realiza una **revisión del código** y envía mensajes al desarrollador en el caso de que surja algún comentario.

7. Si hay algo que corregir, **trabajar los nuevos cambios** y seguir los mismos pasos desde el punto 2.

8. Si está todo correcto, puede que exista alguna persona con el role de "Merger" que acepte los cambios o simplemente algún miembro del equipo realiza la fusión.



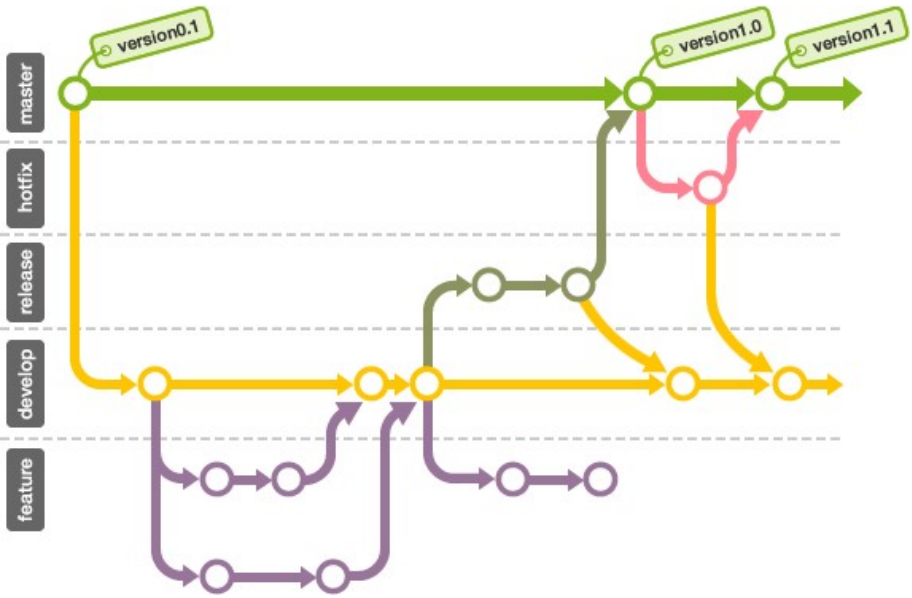
(https://www.paradigmadigital.com/wp-content/uploads/2016/08/Git3.jpg)

Actualización de Stage y Master utilizando Pull Requests

En el momento en el que se decida sacar un nuevo release, puede ser necesario **actualizar las ramas de Stage** (UAT / Release) **o Master** (última versión estable del producto). Para esto, el proceso a seguir consiste en **realizar una solicitud de integración desde Development a Stage o Master**, según corresponda.

La recomendación es primero hacer una solicitud de integración a Stage, en la cual el usuario valide la funcionalidad y luego, en el caso de que la versión sea validada, realizar desde esta rama una solicitud de integración a Master.

Este proceso evita realizar commits directos en Stage o Master y asegura que lo que se está volcando es la diferencia entre ambas ramas, es decir, las nuevas funcionalidades.



Estructura de ramas. Fuente: https://backlogtool.com (https://backlogtool.com)

Beneficios de usar Pull Requests

Desde hace ya algunos años los Pull Request han ganado un lugar en la forma de trabajo de los equipos. Los beneficios que propone se categorizan tanto desde el punto de vista disciplinario como de asegurar la calidad del código que se está produciendo. Algunos de los **beneficios** que se pueden mencionar son:

- **Usar Pull Requests como Code reviews**: La mayoría de las plataformas de versionado de Git permiten hacer Pull Request y generalmente tienen una interfaz amigable para entender el flujo de mensajes entre los distintos desarrolladores. Provee una disciplina de que todo código debe ser revisado. Si bien existen herramientas automáticas que nos pueden solucionar muchos problemas en el código, no hay nada como la revisión de algún miembro del equipo para validar que se utilicen los elementos que ha definido el equipo para el desarrollo (estructuras de código, librerías, estándares, etc).

- **Mejora el codebase del proyecto**: Al hacer revisiones periódicas, el equipo puede validar que el código del proyecto se adapte a los estándares que se han definido.

- **Ayuda a nuevos miembros del equipo** o con poca experiencia a alinearse a los estándares y políticas de creación de código que tenga el equipo. Por su parte, el equipo puede recomendar propuestas de mejora y ajustar aspectos que se consideren necesarios para evitar contaminar la base de código.

- Crear un sitio en donde los desarrolladores pueden generar **pruebas de concepto y compartirlas con el resto del equipo** sin la necesidad de mezclarlo con la versión estable del producto.

- Si una historia de usuario no está terminada o no cumple con el "Definition of Done" es más fácil mantenerla aislada de la versión estable de código.

- Ayuda a **mantener un mejor historial de cambios** evitando comentarios del estilo "WIP", "Tareas hasta hoy", "Backup", etc. Es una buena práctica en general antes de hacer la solicitud de integración, realizar un squash de los commits.

- Definir una correlación entre cada funcionalidad a crear con las ramas generadas en el desarrollo. Si se quiere atacar una funcionalidad en particular es más fácil de encontrar si hay una rama que la representa.

- Es una manera de **mantener un versionado prolijo** tratando de mantener entornos con versiones estables y funcionando.

- Antes de realizar la solicitud de integración puede resultar una buena práctica, correr la batería de tests automáticos del proyecto para validar que una vez que se integre la rama a la de desarrollo, pasará todos los tests definidos. Se trata de **priorizar la prevención**.

Si bien es común pensar que varias de las acciones aquí mencionadas pueden ser resueltas con herramientas automáticas, siempre es positiva una inspección del equipo para asegurar que la calidad del código generado es la mejor posible.

De forma adicional, permite mantener un historial limpio y claro de los cambios que hemos realizado (cosa que no siempre se cuida en proyectos donde trabajan muchas personas) para que el change log del proyecto resulte entendible.

Fundamentalmente, el máximo beneficio, es **lograr que el equipo trabaje en forma mancomunada y colaborativa**, tratando de ayudarse entre ellos a mantener el máximo estándar posible de calidad de código tanto del proyecto como de la empresa donde realizan sus proyectos.

Fuentes

- Git
- Atlassian
- The Backlog Tool 828-2012 – IEEE Standard for Configuration Management in Systems and Software Engineering

Federico Salomone

Soy Ingeniero en Informática con un máster en Dirección Estratégica de la Información. Me apasionan las tecnologías, el desarrollo de software y todo lo que rodee la gestión de un proyecto. Tengo más de 10 años de experiencia en el desarrollo de software en plataformas móviles y web, dirección y liderazgo de proyectos,

gerenciamiento y consultoría. Actualmente trabajo en Paradigma como Scrum Master.

Ver toda la actividad de Federico Salomone

ESCRIBE UN COMENTARIO

Nombre *

Mail (no será publicado) *

Página web

Comentario:

Enviar comentario

En Twitter

@paradigmate
¿Qué ventajas tiene un equipo multifuncional frente a uno que no lo es? ¿Por qué apostar por este tipo de equipos? https://t.co/lhDRc43f0z (https://t.co/lhDRc43f0z)

En nuestro blog

¿Por qué tus equipos deberían ser multifuncionales?
Tras asistir a un taller de Scrum y salir con mil ideas hirviendo en mi cabeza, intenté darle vueltas a cómo mejorar las cosas dentro de mi equipo. Tras oír ...

{ paradigma

91 352 59 42

Vía de las dos Castillas, 33 - Ática 4,
2ª Planta 28224
Pozuelo de Alarcón (Madrid)

newsletter

bimestral
acepto términos legales

contacto download vcard

legal