



UNIVERSIDAD TECNOLÓGICA DE PANAMÁ



FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

LICENCIATURA EN DESARROLLO DE SOFTWARE

ASIGNATURA

DESARROLLO DE SOFTWARE V

PROFESOR

ING. ERICK AGRAZAL

PRESENTACIÓN #1

FLUJOS DE TRABAJOS EN GIT

INTEGRANTES GRUPO #4:

OMAR MONTOYA

8-911-614

AXEL CISNERO

8-812-703

ALEXANDER TORIBIO

8-994-2483

FECHA DE ENTREGA

JUEVES 15 DE AGOSTO DE 2024

1.0 Flujos de Trabajo en Git

1.1. ¿Qué es un flujo de trabajo?

Para el desarrollo de un proyecto se utilizan ramas con un fin específico para desarrollar funcionalidades en conjunto. Al utilizar distintas ramas para fines específicos le damos un espacio separado para que cada desarrollador trabaje en sus implementaciones sin afectar el código que otros estén desarrollando en paralelo.

Una vez finalizadas estas funcionalidades por parte de un desarrollador, su trabajo será unido (merge) con la rama principal o una rama superior. Esto permite que el resto del grupo de desarrollo pueda obtener una versión nueva con los funcionalidades actualizadas. Esto permite desarrollar funcionalidades y también arreglar bugs de manera más sencilla y efectiva.

Un flujo de trabajo (también denominado branching strategy en inglés) es una estrategia que adoptan grupos de trabajo para escribir, combinar e implementar código al utilizar un gestor de versiones (en nuestro caso Git).

Los flujos de trabajos son un conjunto de reglas que los desarrolladores deben seguir para interactuar con las distintas ramas del repositorio de un proyecto.

Estos flujos de trabajo son esenciales para mantener una organización en el repositorio donde se aloja el código del proyecto, evitar errores en la unificación de código y principalmente tener un control sobre muchísimos merge que se harán en el repositorio.

Los flujos de trabajo van de la mano de los procesos DevOps cuya finalidad es mantener siempre un flujo de trabajo rápido y eficiente que garantice el cumplimiento de funcionalidades en un tiempo determinado. Al implementar un flujo de trabajo en nuestro equipo se evitará conflictos entre las modificaciones de cada desarrollador haga sobre el proyecto.

1.2. Importancias y ventajas de un flujo de trabajo

Utilizar un flujo de trabajo nos permite:

- Aumentar la productividad de nuestro equipo de trabajo asegurando coordinación entre desarrolladores.
- Permite desarrollo paralelo de diferentes funcionalidades sobre el mismo proyecto sin afectarse entre sí.
- Permite emplear metodologías DevOps para planificar y estructurar el desarrollo de un proyecto en un tiempo estipulado.
- Los desarrolladores tienen claro el estado del proyecto y en que deben trabajar exactamente.
- Permite notificar a todos los desarrolladores del equipo en qué está trabajando cada uno en un momento determinado.
- En el caso de que se produzcan bugs con el proyecto en producción, los flujos de trabajo nos permiten arreglarlos de manera rápida sin afectar al código de los miembros del equipo.

1.3. ¿Por qué Git para el manejo de flujos de trabajo?

Existen múltiples tecnologías para el control de versiones como Git, Subversion (SVN), Mercurial, Perforce, etc.

¿Por qué Git es la más utilizada? Puede que una de las respuestas a esta pregunta provenga justamente de los flujos de trabajos que ofrece Git y las ventajas que ofrece.

Git tiene una gran capacidad para manejar múltiples ramificaciones y fusiones de manera sencilla y eficiente lo que le permite ser una excelente opción para grandes grupos de trabajo.

Git está orientado a facilitar la colaboración entre desarrolladores y mantener un seguimiento claro de los cambios que se van generando cuando se trabaja sobre un proyecto.

Cuando utilizamos commits Git crea una “foto” del estado actual de tus archivos y guarda una referencia a esa foto. Esto permite que el manejo de ramas sea ligero a diferencia de otros sistemas de control de versiones que guardan literalmente los datos lo que ralentiza el manejo del repositorio y utilizan más espacio de almacenamiento.

Los flujos de trabajo de Git nos permiten utilizar muchas ramas y copias de nuestro código sin utilizar grandes cantidades de almacenamiento. Nos permiten trabajar en proyectos complejos, pero de manera sencilla individualmente.

1.4. Flujos de Trabajo en Git

Dentro de los flujos de trabajo más utilizados en Git tenemos:

1. **Git Flow**
2. **GitHub Flow**
3. **GitLab Flow**

1.4.1. Git Flow

1.4.1.1. Concepto

Git Flow permite el desarrollo de múltiples funcionalidades de manera paralela en equipos de trabajo grandes.

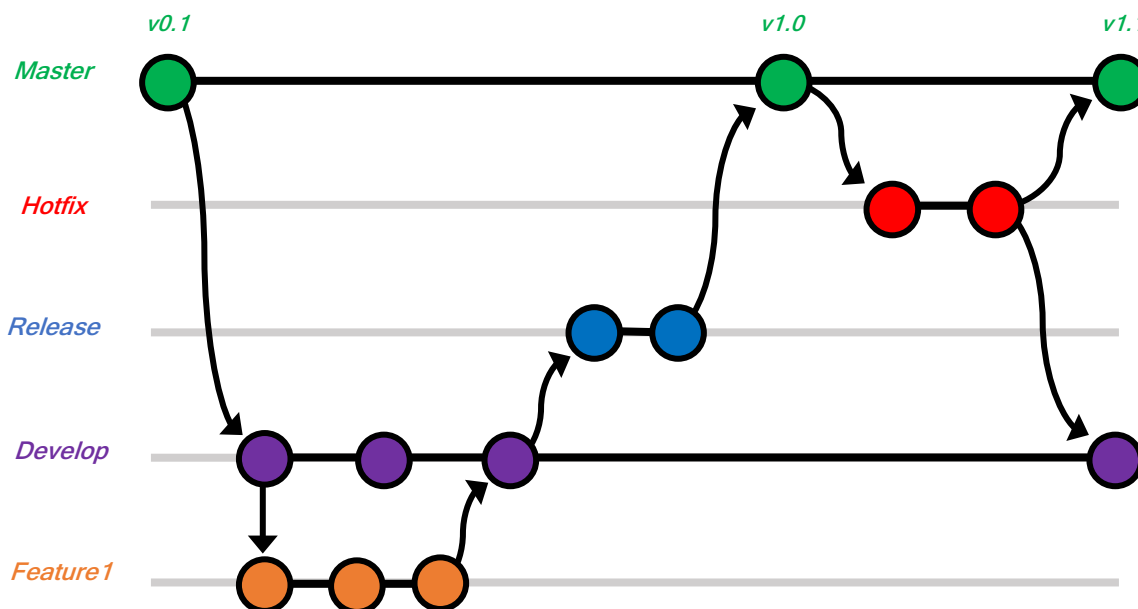
Se crean ramas a partir de la rama master en donde cada desarrollador trabajará en una funcionalidad específica.

Una vez la funcionalidad esté completada, los desarrolladores unirán la rama donde se encuentra la funcionalidad nueva con la rama de desarrollo (mediante un Pull Request).

1.4.1.2. Arquitectura

Este flujo de trabajo consiste en las siguientes ramas:

- **Master**
 - Es la rama principal del repositorio en donde se guardarán las versiones oficiales del proyecto. Es donde estará alojado el código de producción.
- **Develop**
 - A partir de la rama master se crea la rama "develop".
 - Esta rama funciona como una rama intermedia entre las nuevas funcionalidades y el proyecto en producción.
 - Esta rama siempre se mantendrá mientras el proyecto este andando.
- **Feature**
 - Cada funcionalidad nueva debe ser desarrollada en una rama específica derivada de la rama "develop".
 - Una vez terminada el desarrollo de la funcionalidad, se hará un merge desde esta rama hacia la rama "develop".
 - Se pueden tener múltiples ramas "feature" al mismo tiempo.
- **Release**
 - Se utiliza para preparar el código de la rama de develop antes de hacer el merge con la de master/producción.
 - Se genera a partir de la rama de develop.
 - Una vez terminada la preparación, se debe hacer un merge de esta rama tanto con la rama de master y con la rama de develop.
- **Hotfix**
 - En el caso de que se genere un bug en el código de producción, se generará esta rama a partir de master para trabajar de manera rápida sobre el bug y arreglarlo.
 - Permite que los desarrolladores continúen trabajando en implementaciones sin afectar su trabajo en curso.



1.4.1.3. Ventajas de Git Flow

- Forma sencilla de desarrollar en paralelo y proteger el código final de producción.
- Se desarrolla en una rama separada por completo de la rama master.
- Es una manera sumamente organizada para trabajar en equipo gracias a las múltiples ramas.
- Ideal para proyectos que manejan múltiples versiones (versioning) del producto en producción.
- Funciona bien para construcción de software monolito.

1.4.1.4. Desventajas de Git Flow

- Puede llegar a ser complicado para desarrolladores juniors con poca experiencia en el uso de control de versiones.
- Mientras mas ramas sean creadas en el repositorio, el flujo de trabajo puede verse complicado debido a la gran cantidad de commits que se van generando.
- Cuando los test fallan pueden ser complicado encontrar en dónde fallo debido al número de commits.
- Debido a la complejidad de las ramas puede llegar a mermar la rapidez del desarrollo y por ende la entrega de versiones.
- No es recomendable para proyectos que emplean integración continua (*continuous integration*) y entrega continua (*continuous delivery*).

1.4.2. GitHub Flow

1.4.2.1. Concepto

GitHub Flow es una alternativa más simple a Git Flow. Es ideal para proyectos en el que se trabajara con equipos pequeños que no necesitan gestionar múltiples versiones del mismo proyecto en producción.

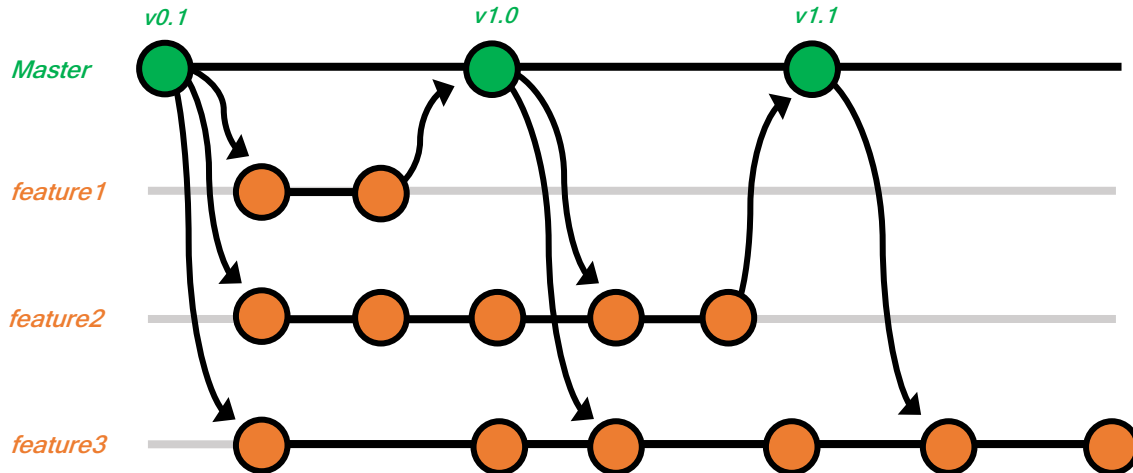
Este modelo no posee una rama de "release". Solo se crean las ramas de desarrollo de funcionalidades directamente desde la rama master para luego incorporarlas de vuelta en la rama master.

Debido a su simpleza de ramas es muy utilizado para proyectos que necesitan desplegar su código constantemente por ende es ideal para la integración y entrega continuas.

1.4.2.2. Arquitectura

- **Master**
 - Es la rama principal del repositorio en donde se guardarán las versiones oficiales del proyecto.
 - Es donde estará alojado el código de producción.
 - Se mantendrá en constante desarrollo para el "deploy" de versiones.
- **Feature**
 - Cada funcionalidad será creada en una rama distinta.

- Los desarrolladores podrán trabajar paralelamente y una vez terminado su desarrollo harán Pull Request para hacer merge entre su rama y la rama master.
- Los test se hacen directamente en las ramas de funcionalidades.



1.4.2.3. Ventajas de GitHub Flow

- Útil para metodologías ágiles para el desarrollo rápido con tiempos de producción pequeños y entrega de versiones continuas.
- Las revisiones de código se hacen de manera rápida para que modificaciones se realicen de inmediato.
- Poca burocracia entre ramas ya que esencialmente solo hay dos tipos: Master (producción) y feature (desarrollo de funcionalidades). Esto agiliza el proceso de desarrollo.
- Ideal para equipos pequeños de desarrollo.
- Ideal para programas que mantienen solo una versión en producción.

1.4.2.4. Desventajas de GitHub Flow

- No es útil en proyectos donde se debe mantener múltiples versiones de la aplicación.
- Debido a que solo se tienen dos ramas es más susceptible a generar bugs por lo que las versiones en producción pueden llegar a ser inestables.
- La rama master puede llegar a estar sobresaturada ya que se utiliza como la rama de producción y de desarrollo al mismo tiempo.
- Es más difícil que los desarrolladores estén claros en que funcionalidad está trabajando otros miembros del equipo.

1.4.3. GitLab Flow

1.4.3.1. Concepto

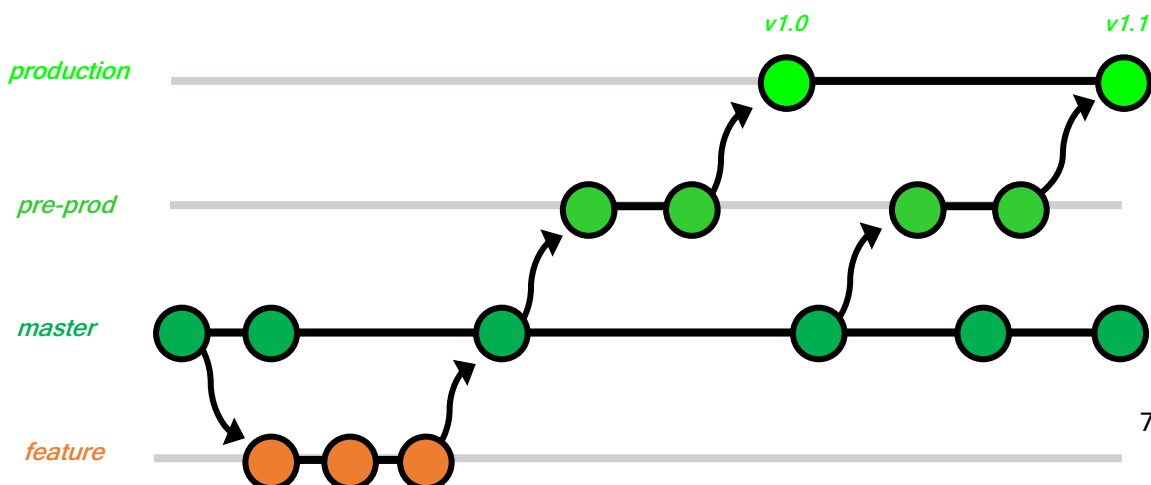
Es una alternativa simple a Git Flow que emplea el desarrollo enfocado a funcionalidades y también utiliza ramas para cada funcionalidad.

En GitLab se trabaja directamente sobre la rama master, a diferencia de Git Flow que se trabaja en la rama de develop para luego ir a master.

Permite el mantenimiento de múltiples entornos. Los utilizan los equipos de trabajo que necesitan tener un entorno de "staging" (master branch) separado del entorno de producción (production branch). Una vez la rama master esta lista con una funcionalidad nueva pasa a la rama de pre-producción para luego ir a producción.

1.4.3.2. Arquitectura

- **Feature**
 - Son las ramas en donde los desarrolladores trabajarán las nuevas funcionalidades.
 - Se deriva desde la rama master
- **Master**
 - Es la rama que recopila todas las funcionalidades que los desarrolladores han trabajado.
 - A diferencia de Git Flow o GitHub Flow en la que la rama master siempre tendrá el código listo para producción, en GitHub Flow el código de la rama master aún debe pasar por otros entornos para estar listo para producción.
- **Pre-Production**
 - Consiste en entornos intermedios entre master y producción.
 - Se utiliza para arreglar realizar tests y limpiar el código de master.
 - Los equipos de trabajo pueden crear varias ramas de pre-produccion para testear distintos entornos.
- **Production**
 - Es la rama donde está el código final de producción.
 - En algunos casos pueden utilizarse para mantener distintas versiones en ramas distintas de producción haciendo posible que se mantenga cada una de manera individual.



1.4.3.3. Ventajas de GitLab Flow

- Permite trabajar con equipos grandes de trabajo y a la vez permite integración continua.
- La utilización de ramas de preproducción y producción asegura que el producto final esté libre de errores.
- Permite darle mantenimiento a distintas versiones del producto en producción y mantenerlas simultáneamente.
- Útil para proyectos en el que no se puede controlar los tiempos de desarrollo (Aplicaciones móviles, por ejemplo).

1.4.3.4. Desventajas de GitLab Flow

- A comparación con GitHub Flow es más complicado y el código debe pasar por muchas ramas hasta llegar a producción.
- Puede llegar a ser bastante complicado de aprender para equipos pequeños o sin experiencia en la gestión de versiones.
- Para que funcione de manera óptima se debe configurar el continuous integration y el continuous delivery.

2.0. Pull Requests

2.1. ¿Qué es un Pull Request?

Un pull request es una herramienta que facilita que desarrolladores colaboren en un proyecto utilizando Git.

Un pull request (*también conocidos como "PR" dentro de la comunidad de desarrolladores*) funciona como un mecanismo para notificar a todos los miembros del equipo de desarrollo que un nuevo "feature" o funcionalidad ha sido completada por parte de un colaborador y se desea agregar esta funcionalidad al proyecto final (usualmente la rama *master* o *develop*).

Lo conveniente de trabajar en equipo y generar pull requests entre desarrolladores y el personal encargado de administrar el proyecto es que con cada Pull Request se pueden revisar y analizar entre múltiples usuarios los cambios propuestos antes de integrarlos directamente en el proyecto final. De esta manera se pueden evitar "bugs" o problemas de implementación que podrían romper el producto final de no ser revisados antes.

En algunos proyectos se podrían tener miles y miles de líneas de código y con los pull requests se pueden mostrar fácilmente las diferencias entre el código nuevo de la rama fuente (desde donde se hizo el pull Request) y el código de la rama destino (hacia donde se desea enviar los cambios).

2.2. Estructura de un Pull Request

Cuando haces un PR estas pidiendo que otro desarrollador haga un *pull* a una rama que se encuentra en tu repositorio (*source branch*) hacia una rama que se encuentra en su repositorio (*destination branch*).

Para hacer un PR se necesita tomar en cuenta cuatro elementos:

1. Repositorio destino o final:

- Representa el repositorio oficial del proyecto en el que se esté trabajando conjuntamente. Es el repositorio donde todos los desarrolladores consolidan sus trabajos.
- Usualmente le pertenece al propietario del proyecto o a los mantenedores asignados.

2. Rama destino o final:

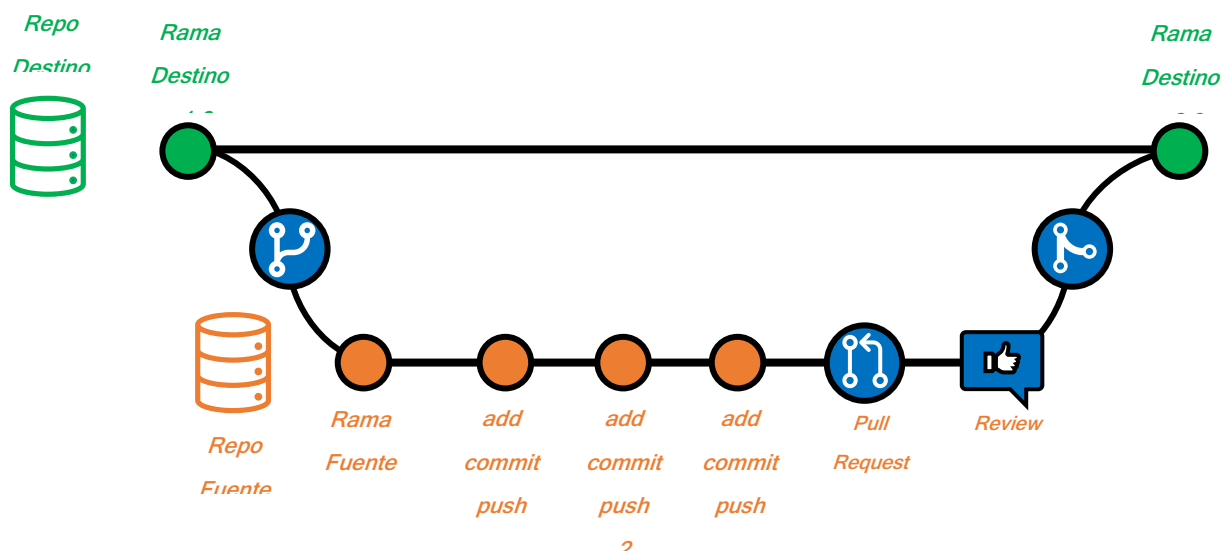
- Dependiendo del flujo de trabajo que se esté utilizando tendría diferentes nombres. Usualmente sería la rama "master", "developer", "feature", "hotfix".
- Representa la rama donde se encuentra todo el código en el que el equipo se encuentra colaborando conjuntamente.
- Cualquier cambio en esta rama será notificado a todo el equipo de trabajo para asegurar que se esté trabajando con código actualizado.
- Nuestra finalidad es colaborar en el proyecto y una vez finalizado integrar nuestro código con el de esta rama para que todo el equipo lo utilice.

3. Repositorio fuente o inicial:

- Es el repositorio del cual **proviene los cambios** que se estén proponiendo en el pull request para ser integrados con el repositorio destino.
- En este repositorio se encuentra la **rama fuente** que contiene el código en el que se ha trabajado con las nuevas funcionalidades.
- Se pueden presentar dos casos en base a la naturaleza del proyecto:
 1. **Proyecto privado de un equipo dentro de una empresa:**
 - Si se trabaja en una empresa lo mas probables es que el repositorio fuente y el repositorio destino sean el mismo.
 - Básicamente siempre se estará trabajando directamente en el mismo repositorio que el repositorio destino, ya que formamos parte del equipo de trabajo de desarrollo.
 2. **Proyecto Open Source o código abierto:**
 - Para proyectos de código abierto en el que cualquier persona puede contribuir, se suele "forkear" (hacer un "fork") al repositorio destino.
 - Cuando se hace un "fork" el desarrollador obtiene una copia personal del repositorio para que pueda trabajar en sus colaboraciones.
 - A pesar de que el repositorio es una copia exacta del repositorio destino, no son el mismo repositorio ya que el "fork" crea la copia remota del repositorio en el perfil del desarrollador en la plataforma que este utilizando.

4. Rama fuente o inicial:

- Representa la rama en la que el desarrollador podrá escribir su código nuevo y manipular el proyecto a su antojo sin afectar directamente la rama destino.
- Es la rama de donde proviene el Pull Request y que contiene los commits que se agregarán a la rama destino.
- Se crea con un fin específico para desarrollar alguna tarea o funcionalidad particular.



2.3. Funcionalidades de un Pull Request

Los pull requests tienen las siguientes funcionalidades:

- **Notificaciones:**
 - Luego de generar un PR, la plataforma que se esté utilizando se encarga de generar **notificaciones automáticas** a:
 1. Miembros del equipo de desarrollo.
 2. Encargados de revisar el código.
 3. Personas que estén observando el repositorio.
 - Estas notificaciones pueden generarse directamente en la **interfaz de la plataforma** que se esté utilizando y además por **correo electrónico**.
 - Por parte del usuario que genera el PR, éste puede incluir **comentarios y mensajes personalizados** junto con la solicitud para proporcionar más información y contexto de los cambios generados en el PR.
 - Las notificaciones son vitales para un flujo de trabajo efectivo ya que se generarán múltiples cambios en distintas ramas del repositorio, por lo que cada desarrollador debe estar anuente a cada cambio y asegurarse de trabajar en la versión más actualizada del proyecto.
- **Discusiones (comentarios) :**
 - La funcionalidad del pull request va más allá de solo notificaciones. Funciona también como un **foro** para discutir cualquier funcionalidad o solución propuesta.
 - La funcionalidad de comentarios son una herramienta clave para la colaboración y revisión de código en los flujos de trabajo de un equipo de desarrollo.
 - En un PR podemos encontrar **tres tipos de comentarios**:
 1. **Comentarios generales:**
 - Son comentarios que encontramos en el PR para todo el equipo de trabajo.
 - Son utilizados para explicar aspectos generales y dar contexto del PR.
 - También se pueden usar para hacer preguntas abiertas sobre los cambios propuestos en el PR.
 2. **Comentarios de líneas de código:**
 - Son comentarios que se hacen específicamente sobre líneas de código en el PR.
 - Se agregan directamente sobre las línea de código que se están viendo afectadas por los nuevos cambios.
 - Se pueden utilizar para indicar problemas, errores, sugerir cambios, mejoras o simplemente hacer una pregunta sobre el código.
 3. **Comentarios de revisión:**
 - Los encargados de revisar los PR pueden aprobar, solicitar cambios o rechazar un PR utilizando comentarios específicos para cada acción.

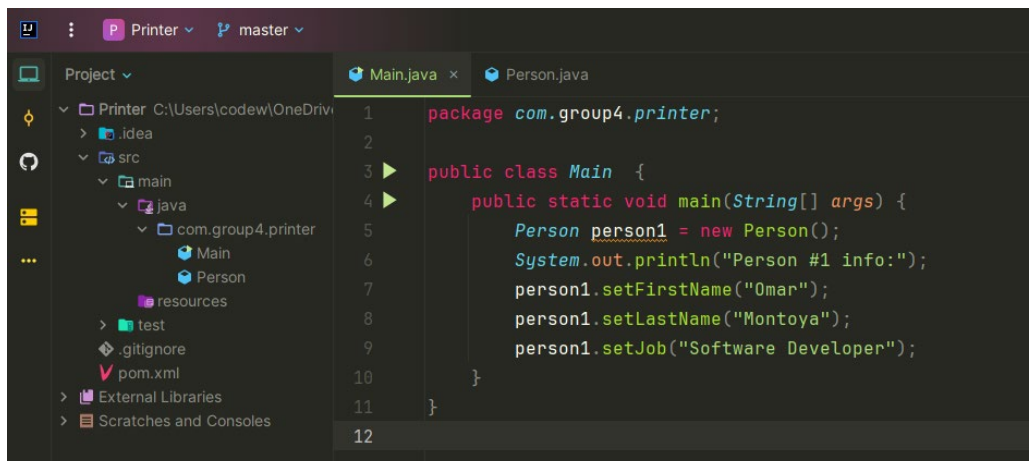
- Estos comentarios contienen feedback sobre lo que se debe corregir, si el código está bien o la razón por la que no se aprobaron los cambios.
- **Follow-Up Commits:**
 - Luego de que se hace una revisión a un PR, el que generó el PR puede generar follow-up commits para implementar las correcciones o mejoras solicitadas por los revisores.
 - También se utiliza para agregar o mejorar la documentación sobre los cambios propuestos en el PR.
 - Otras veces puede que el código no tenga errores, pero se sugieren optimización de código. Se puede usar follow-up commits para implementar tales optimizaciones.
 - La utilización de follow-up commits tiene algunas ventajas ya que facilita mantener el historial del cambios dentro del mismo PR sin la necesidad de crear uno nuevo. Además, la próxima revisión se enfocará solamente en los cambios solicitados y no sobre todo el PR original.

2.4. ¿Cómo hacer un Pull Request?

La metodología detrás de cómo utilizar los pull requests depende del flujo de trabajo que se esté empleando en el grupo de desarrollo. Pero todos los pull requests necesitan dos repositorios o dos ramas distintas del mismo repositorio.

2.4.1. Pull Request en GitHub

- Para nuestro ejemplo utilizaremos un repositorio privado llamado "ds5_group4" ubicado en: https://github.com/codewithomarm/ds5_group4.
- El administrador y dueño del repositorio es el usuario *codewithomarm* (Omar Montoya).
- El usuario *axelcisnero* (Axel Cisneros) fue agregado como parte del equipo de trabajo para que posea todos los permisos necesarios para trabajar en el proyecto y solicitar un PR.
- El proyecto consiste en un programa Java de muestra que imprime la información de una persona en consola:

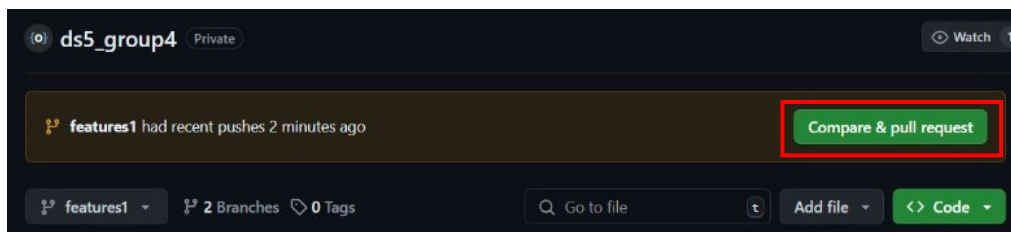


```
1 package com.group4.printer;
2
3 public class Main {
4     public static void main(String[] args) {
5         Person person1 = new Person();
6         System.out.println("Person #1 info:");
7         person1.setFirstName("Omar");
8         person1.setLastName("Montoya");
9         person1.setJob("Software Developer");
10    }
11 }
12
```

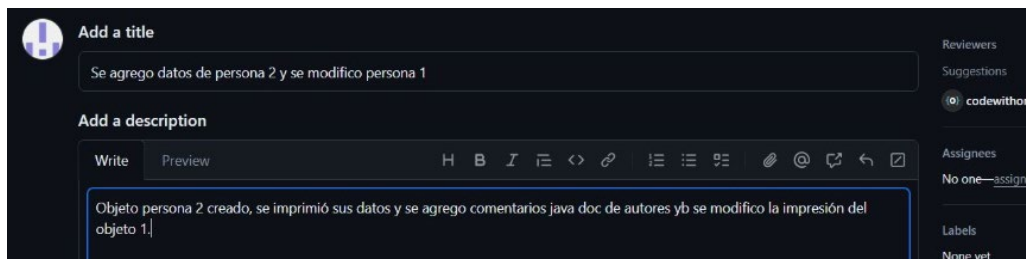
- Se desea que Axel cree una rama "feature1" modifique el archivo Main.java y agregue la información de la segunda persona que se imprimirá en consola.
- Para lograrlo Axel clonó el repositorio (git clone), creó la rama features1 (git checkout), modificó el archivo Main.java, ejecutó los comandos "add", "commit" y "push" para mandar los archivos modificados de su repo local al remoto.
- Una vez completado estos pasos estamos listos para generar nuestro Pull Request.

Pasos para solicitar el Pull Request

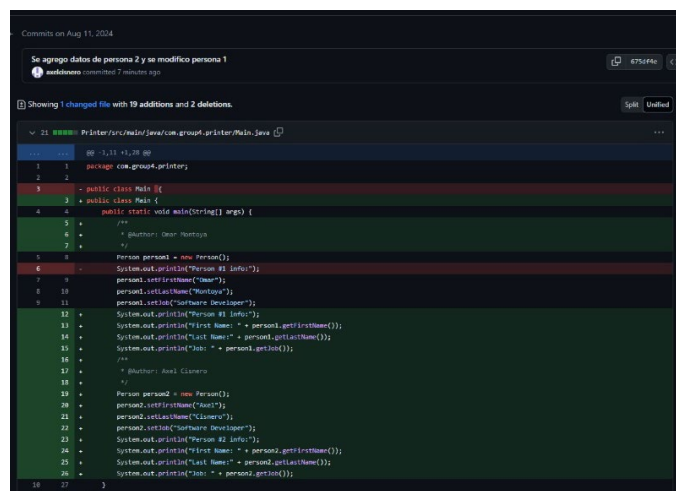
1. Con el usuario desde donde estamos trabajando en la rama inicial, debemos generar la solicitud del Pull Request.
2. Si entramos al repo remoto veremos que GitHub nos indica que debemos solicitar el Pull Request. Hacemos click en "Compare & pull request".



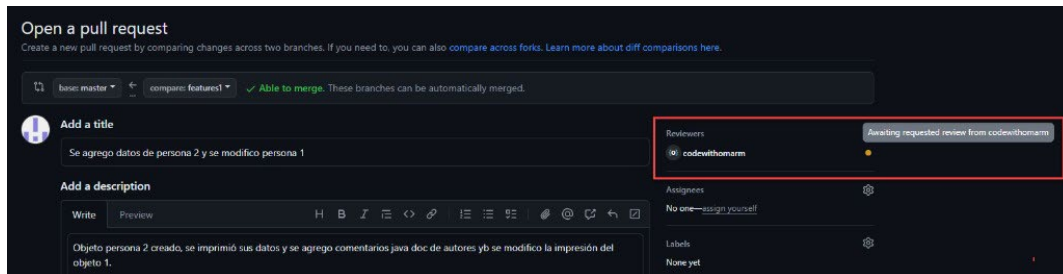
3. GitHub nos mostrara un formulario para agregar un título al PR y agregar un comentario junto con la solicitud.



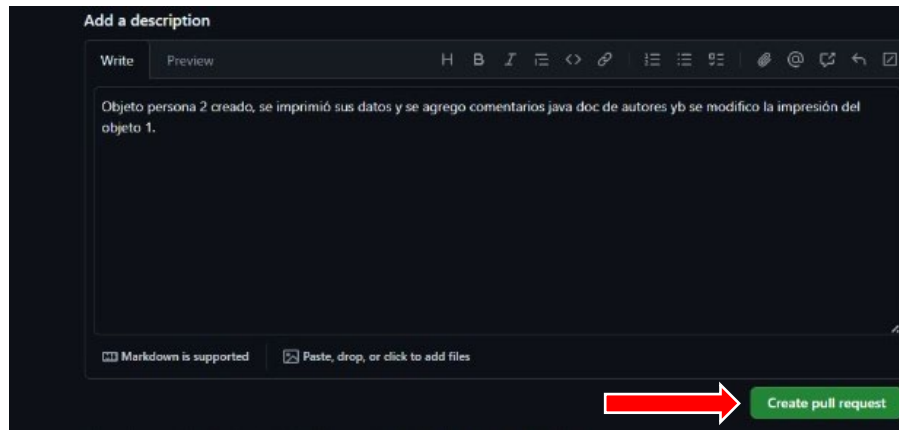
4. Además, en la parte inferior se nos listará claramente las líneas de código que han sido afectadas por el PR.



5. En las opciones de configuración en la derecha de la página podemos ver la sección “reviewers”. Aquí podemos seleccionar los miembros del equipo de deseamos revisen nuestro Pull Request antes de hacer el merge con la rama destino. Damos click en “Request” al lado del usuario que revisará nuestro PR. Una vez seleccionado, GitHub nos confirmará que se está esperando revisión por parte de ese usuario.



6. Para finalizar, Hacemos click en el botón “Create Pull Request” justo debajo de la caja de comentario.



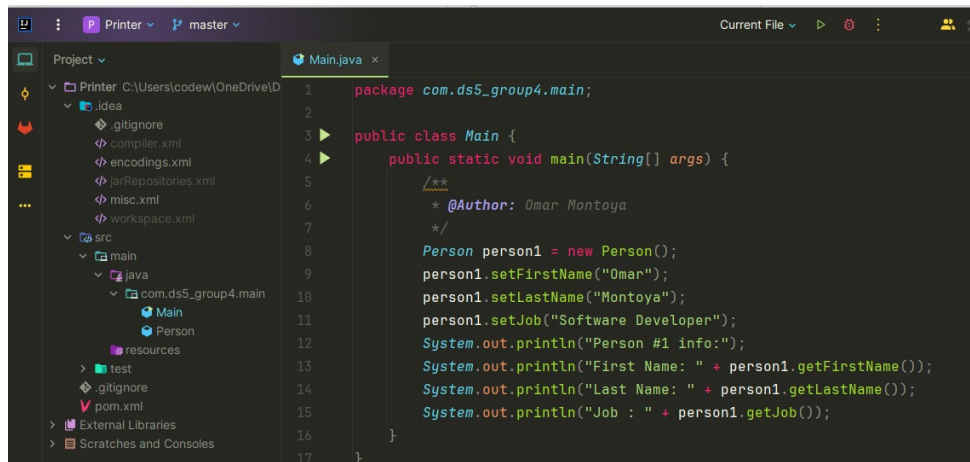
Una vez completado estos pasos nuestro Pull Request estará completado.

Dependiendo del flujo de trabajo que se este utilizando dirá las acciones que se tomarán a continuación.

Los miembros del equipo que escogimos podrán revisar nuestro código, dejar comentarios generales, comentarios de líneas de código y aprobar o declinar nuestro pull request. En algunos casos la revisión puede pedir que realicemos cambios antes de hacer el merge con la rama destino.

1.4.2. Pull Request en GitLab

- Para nuestro ejemplo utilizaremos un repositorio privado llamado “pullrequest” en el proyecto “ds5_group4” ubicado en: https://gitlab.com/ds5_group4/pullrequest.
- El administrador y dueño del repositorio es el usuario *codewithomarm* (Omar Montoya).
- El usuario *alex50789* (Alexander Toribio) fue agregado como parte del equipo de trabajo para que posea todos los permisos necesarios para trabajar en el proyecto y solicitar un PR.
- El proyecto consiste en un programa Java de muestra que imprime la información de una persona en consola:



```
1 package com.ds5_group4.main;
2
3 public class Main {
4     public static void main(String[] args) {
5         /**
6          * @Author: Omar Montoya
7          */
8         Person person1 = new Person();
9         person1.setFirstName("Omar");
10        person1.setLastName("Montoya");
11        person1.setJob("Software Developer");
12        System.out.println("Person #1 info:");
13        System.out.println("First Name: " + person1.getFirstName());
14        System.out.println("Last Name: " + person1.getLastName());
15        System.out.println("Job : " + person1.getJob());
16    }
17 }
```

- Alexander clonó el repositorio (`git clone`), creó la rama `features1` (`git checkout`), modificó el archivo `Main.java`, ejecutó los comandos “`add`”, “`commit`” y “`push`” para mandar los archivos modificados de su repo local al remoto.
- Una vez completado estos pasos estamos listos para generar nuestro Pull Request.

Pasos para solicitar el Pull Request

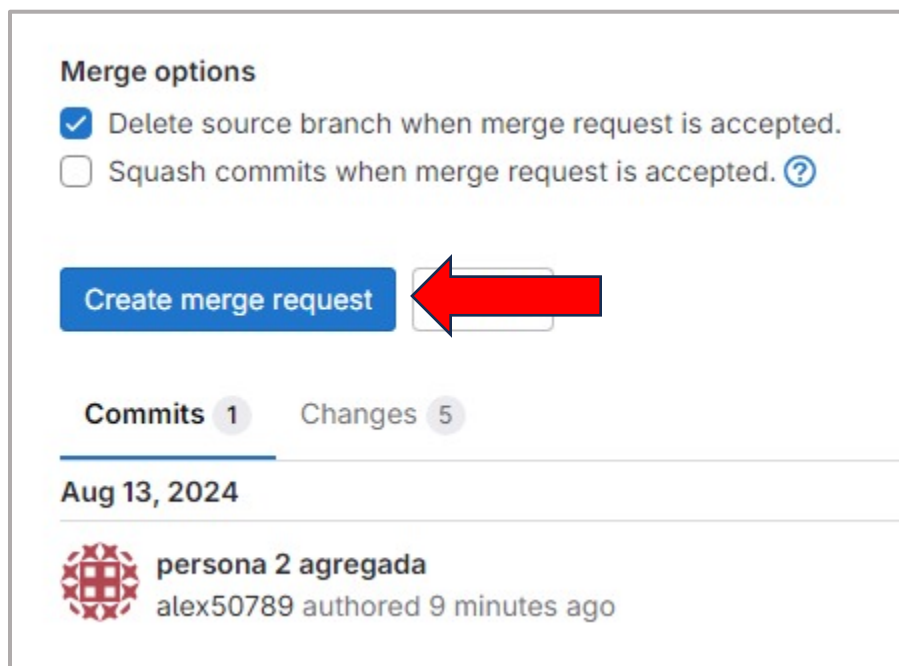
1. Con el usuario desde donde estamos trabajando en la rama inicial, debemos generar la solicitud del Pull Request. En GitLab se llama Merge Request.
2. Si entramos al repo remoto veremos que GitLab nos indica que debemos solicitar el Merge Request. Hacemos click en “Create Merge Request”.



3. Agregamos el título del Merge Request y agregamos un comentario dando contexto del PR. Antes de completarlo debemos seleccionar a los miembros que deben revisar nuestro merge request.



4. Hacemos click en "Create Merge Request" para completar nuestro Pull Request en GitLab



Una vez completado el Pull Request (Merge Request en GitLab) los encargados de revisar le llegarán las notificaciones a su cuenta de GitLab para que revisen nuestro merge request y den su feedback para aprobar, pedir cambios o declinar nuestro merge request.

Conclusión

Axel Cisnero

La implementación de flujos de trabajo en Git es esencial para cualquier equipo de desarrollo que busque eficiencia y coordinación en la gestión de código. Git, al ser una de las herramientas más robustas y flexibles para el control de versiones, permite a los equipos trabajar en múltiples funcionalidades de manera simultánea sin comprometer la estabilidad del proyecto principal. Este nivel de organización es clave para proyectos complejos, donde la colaboración y la integración continua son fundamentales para el éxito.

Además, cada flujo de trabajo ya sea Git Flow, GitHub Flow, o GitLab Flow, ofrece soluciones específicas que se ajustan a diferentes tipos de proyectos y equipos. Git Flow, por ejemplo, es ideal para proyectos grandes y complejos con múltiples versiones, mientras que GitHub Flow es más adecuado para proyectos que requieren un desarrollo rápido y lanzamientos frecuentes. Por otro lado, GitLab Flow combina características de ambos, permitiendo un control más detallado sobre los entornos antes de que el código llegue a producción.

El uso de pull requests no solo ayuda a la revisión y mejora del código, sino que también permite la colaboración entre los desarrolladores. A través de los pull requests, los equipos pueden asegurarse de que cada cambio propuesto sea revisado y ajustado antes de ser añadido al proyecto.

Bibliografía

Acerca de las solicitudes de incorporación de cambios - Documentación de GitHub. (s. f.).

GitHub Docs. <https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>

Atlassian. (s. f.). *Pull Requests* | *Atlassian Git Tutorial*.

<https://www.atlassian.com/git/tutorials/making-a-pull-request>

GeeksforGeeks. (2024, 26 febrero). *Branching strategies In Git*. GeeksforGeeks.

<https://www.geeksforgeeks.org/branching-strategies-in-git/>

GitLab. (s. f.). *What is GitLab Flow?* | *GitLab*. GitLab.

<https://about.gitlab.com/topics/version-control/what-is-gitlab-flow/>

Haddad, R. (2024, 22 julio). *What Are the Best Git Branching Strategies*. Abtasty.

<https://www.abtasty.com/blog/git-branching-strategies/>

Sentrio. (2024, 17 julio). *Herramientas de control de versiones en DevOps*. Sentrio.

<https://sentrio.io/blog/herramientas-control-de-versiones-devops/>