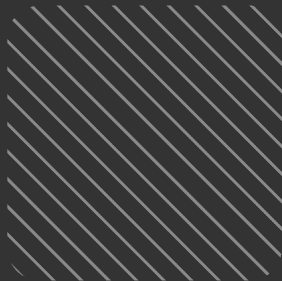


Quality

IT BOARDING

BOOTCAMP



Índice



01

Release process

02

Branching
model

03

Continuous
Integration

04

Code
Coverage

05

Versiones y
Semver

06

Code Review

IT BOARDING

BOOTCAMP

// **RELEASE PROCESS**

IT BOARDING

BOOTCAMP

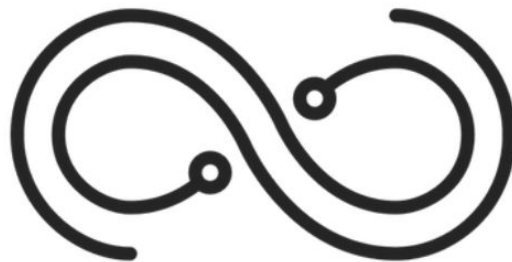
// ¿Qué es Release Process?

Release Process es un **framework** construido íntegramente sobre **Fury** que nos permite como developers, integrar código en las aplicaciones de MeLi de manera fácil, rápida y sobre todo segura.

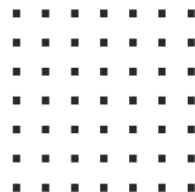
Persigue **garantizar** un **alto estándar de calidad** sobre el proceso y el código que está en juego.

Se apalanca en los conceptos de **Integración Continua** y **Delivery Continuo** y hace uso de muchas herramientas (tanto Open Source como construidas por el equipo) para garantizar un flujo simple, rápido y adaptado a las necesidades que tenemos como equipo de IT dentro de MeLi.

Release Process se ejecuta principalmente en los **Pull Requests** de nuestras aplicaciones



Release Process





¿Qué es un Pull Request?

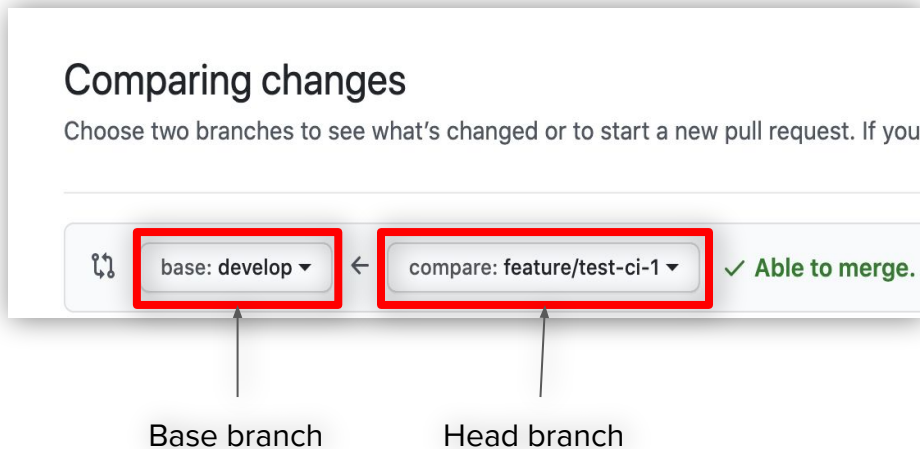
La traducción directa sería algo así como “**Petición de Validación**”.

Los pull request permiten no solo llevar de forma más ordenada las tareas en la etapa del desarrollo, sino también **crear propuestas o cambios que puedan ser integrados posteriormente a dicho proyecto**.

Cada Pull Request pide introducir e integrar cambios **desde** un branch **hacia** otro.

El branch desde donde tenemos los cambios y queremos integrar se le da el nombre de **head branch**.

El branch hacia donde queremos integrar nuestros cambios se lo conoce como **base branch**.



// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

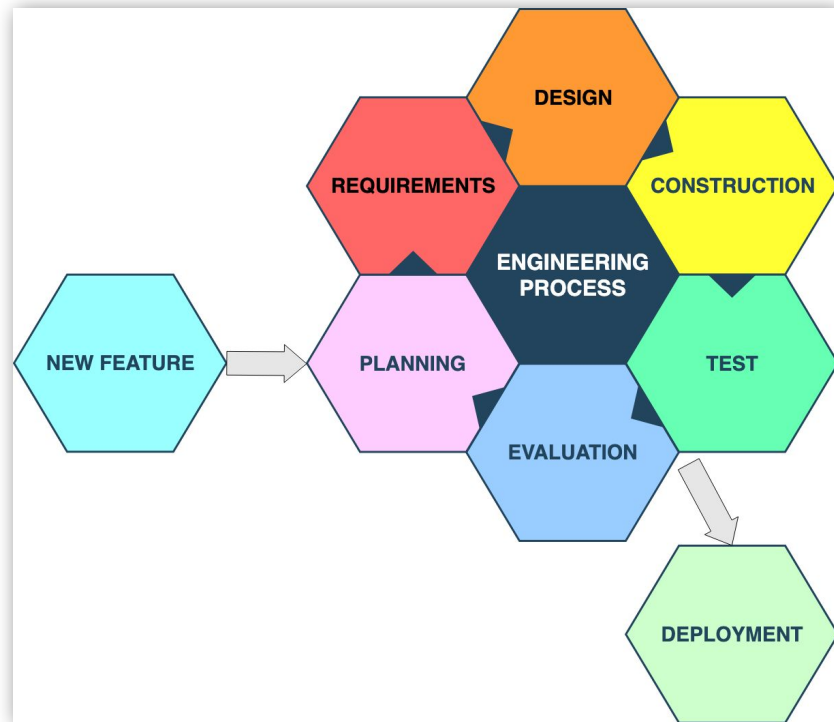
¿Para qué sirve Release Process?

Release Process toma relevancia en el **proceso de Ingeniería de Software** que realizamos como desarrolladores en nuestro día a día.

En ese sentido, el uso de Release Process forma la **experiencia de desarrollo (DX)** que se busca estandarizar en todo MeLi.

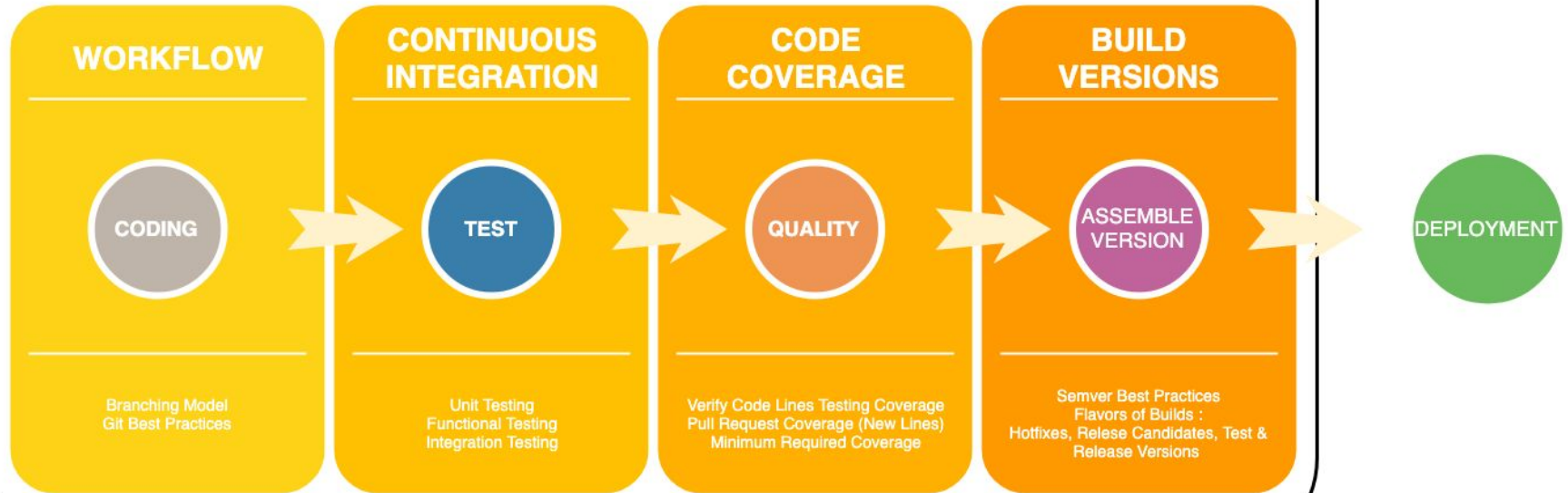
Entre sus responsabilidades:

- Crea un proceso unificado, auditable y trazable para cada cambio que se realiza en las aplicaciones.
- Garantiza un alto nivel de calidad en cada porción de código introducido.
- Agiliza el proceso de empaquetado y despliegue de las aplicaciones.



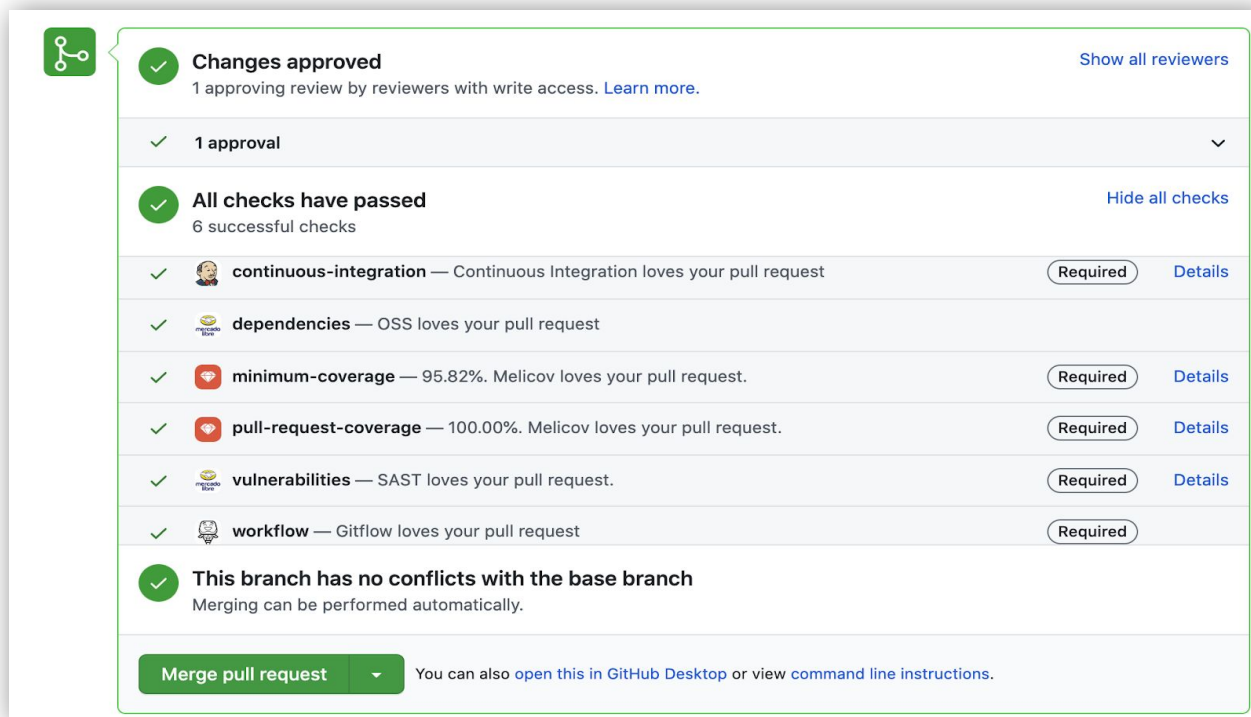



CONSTRUCTION & TEST PHASES




Cómo vemos lo que ejecuta Release Process







Podemos verificar lo que ejecuta release process en los Pull Requests de nuestro repositorio. Se identifican cómo un conjunto de checks y validaciones que se visualizan dentro del mismo y dan visibilidad al desarrollador del resultado de cada una de las etapas. A continuación se explicará cada una de las etapas.




 **Changes approved** [Show all reviewers](#)
1 approving review by reviewers with write access. [Learn more.](#)

✓ **1 approval** 

✓ **All checks have passed** [Hide all checks](#)
6 successful checks

- ✓  **continuous-integration** — Continuous Integration loves your pull request Required [Details](#)
- ✓  **dependencies** — OSS loves your pull request
- ✓  **minimum-coverage** — 95.82%. Melicov loves your pull request. Required [Details](#)
- ✓  **pull-request-coverage** — 100.00%. Melicov loves your pull request. Required [Details](#)
- ✓  **vulnerabilities** — SAST loves your pull request. Required [Details](#)
- ✓  **workflow** — Gitflow loves your pull request Required

✓ **This branch has no conflicts with the base branch**
Merging can be performed automatically.

[Merge pull request](#)  You can also [open this in GitHub Desktop](#) or view [command line instructions](#).



// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

// **BRANCHING MODEL**

IT BOARDING

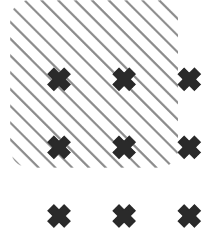
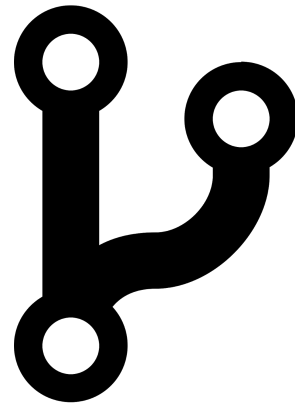
BOOTCAMP

// ¿Que es un Branching Model y para qué sirve?

Como programadores, rara vez trabajamos en solitario en un proyecto. Lo normal es que nos acompañen otros desarrolladores que colaborarán con nosotros mano a mano. A veces nos encontramos con situaciones como tener que trabajar en funcionalidades nuevas, arreglar bugs críticos, acabar una iteración y tener que juntar todas las funcionalidades que se han desarrollando hasta el momento, o incluso dejar lo que estábamos haciendo para seguir con otra tarea más importante o probar implementaciones.

Estas situaciones descritas anteriormente pueden llegar a ser un gran problema si un equipo de desarrollo carece de algún sistema de organización en su repositorio de control de versiones.

Los “branching models” son esquemas de trabajo en repositorios donde se describe el **flujo de desarrollo** por el que debe pasar cada nuevo desarrollo.



En MeLi utilizamos dos branching models:

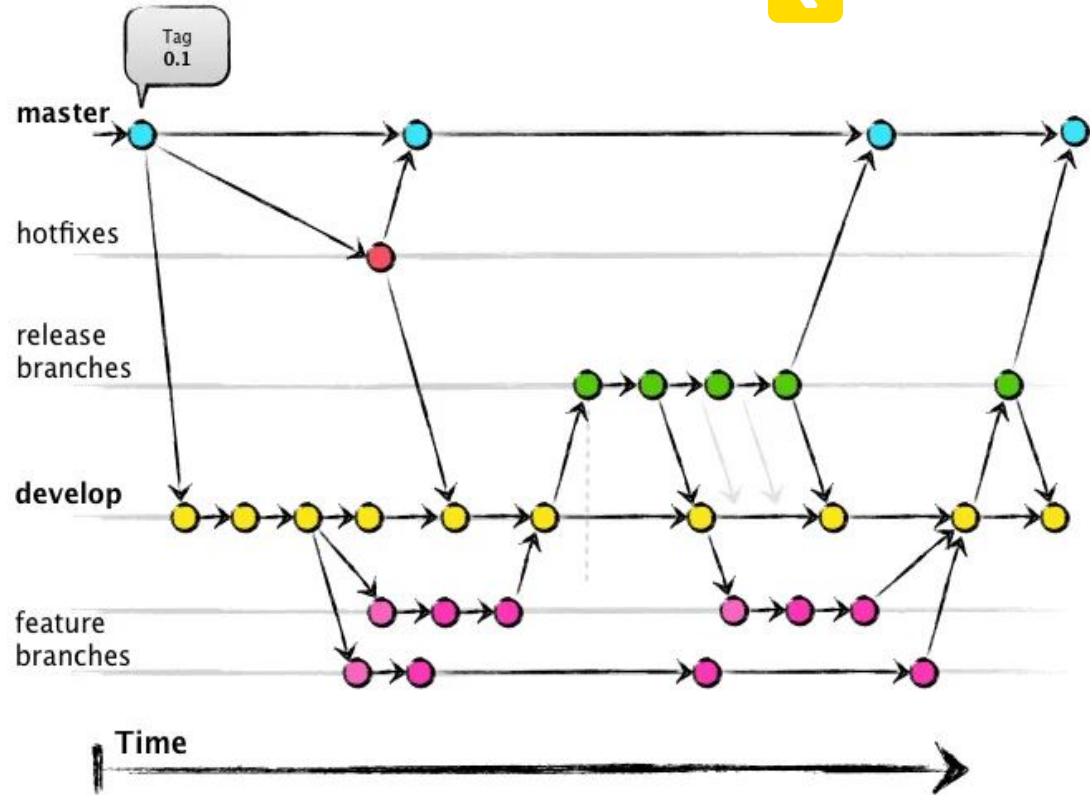
- **Gitflow**
- **Libflow** (Especifico para Librerías)

Al usar estos workflows estandarizados en MeLi, todos los desarrolladores pueden cambiar de proyecto y aplicaciones pero el flujo de trabajo se mantiene. No hay necesidad de adaptarse a nuevas formas de trabajo en cada aplicación distinta.



Características:

- Dos branches principales (estables y protegidos)
 - master
 - develop
- Los nuevos desarrollos se trabajan desde branches que comiencen con nombre **feature/** y se abren pull requests hacia **develop** para integrar.
- Cuando se desea crear una versión, a partir de develop se crea un branch **release/** con el número de la versión que se desea crear. Por ejemplo release/1.0.0
- La versión y tag final se crea una vez que se realiza el merge del branch **release/ a master**
- En caso de existir un bug crítico, es posible crear un branch **hotfix/** directamente apuntando hacia master sin haber integrado a develop previamente.



// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

// ¿QUÉ ES CONTINUOUS INTEGRATION?

IT BOARDING

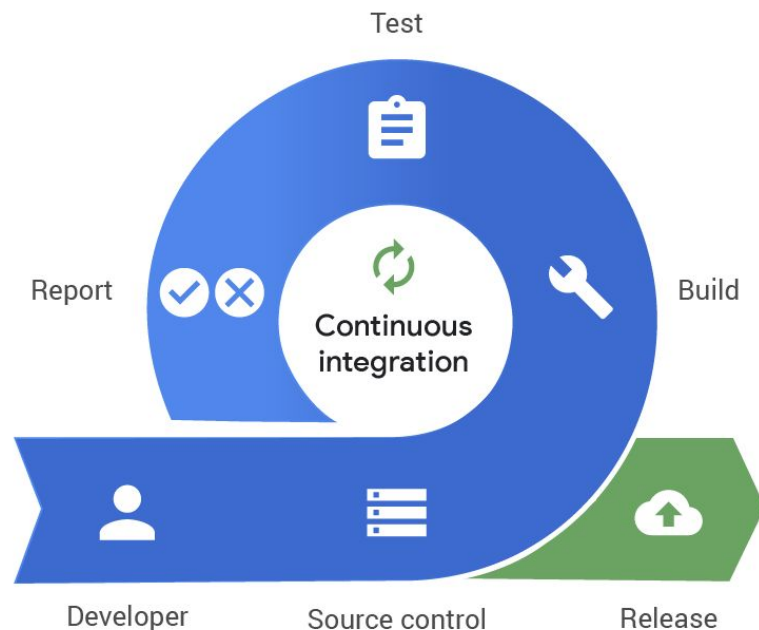
BOOTCAMP

// Continuous Integration

La integración continua se refiere al proceso por el cual se efectúan una serie de validaciones que se deben respetar para integrar el código desarrollado a una aplicación.

Beneficios:

- **Se establecen y validan las condiciones necesarias para que el código pueda ser integrado**
- **Detección de error de forma temprana y rápida**
- **Mayor eficiencia del trabajo en equipo.**
- **Asegura la integridad del código.**



// ¿CÓMO SE USA?

IT BOARDING

BOOTCAMP

// ¿Cómo se usa Integración Continua?

Todos pueden aportar algo en el Code Review. **Para hacerlo deberíamos tener en cuenta algunos puntos como:**

1. **Desarrollo:** El desarrollador trabaja localmente sobre el desarrollo que esté llevando a cabo.
2. **Subida a Github:** Una vez realizado el trabajo localmente, el desarrollador debe subir sus cambios. Para esto usamos el VCS **Git**, a través del cual se subiran sus cambios a una rama (**branch**) propia.
3. **Ejecución del Integrador Continuo:** Un integrador continuo puede ejecutarse en base a diferentes eventos. El más común es ejecutarlo cuando existe un pedido de integración (**Pull Request**) en nuestro repositorio remoto en Github. Este evento sirve de disparador para ejecutar el integrador continuo.
4. **Fases del Integrador Continuo:** Una vez iniciada la ejecución del integrador continuo, se ejecutarán todas las validaciones necesarias para asegurar la calidad del código.
5. **Resultado:** Se obtendrá un resultado positivo o negativo dependiendo si las validaciones fueron cumplidas satisfactoriamente. Este resultado será visible en el Pull Request



Los resultados de la ejecución de Integración Continua permiten tener una **mejor visibilidad** de los desarrollos de un repositorio y **facilita la toma de decisiones** en base a los resultados obtenidos

IT BOOTCAMP | **QUALITY**

// **CI EN FURY**

IT BOARDING

BOOTCAMP

// CI en Fury

El proceso de Integración Continua en Fury se provee como un servicio listo para ser utilizado en cualquier aplicación dentro de Fury. **No se requiere ninguna configuración manual por parte de los desarrolladores**

El proceso se ejecuta:

- Al abrir un Pull Request
- Al realizar un commit sobre un branch con Pull Request abierto

The screenshot displays a GitHub Pull Request titled "First code introduced #1". The pull request is from the `feature/first-code` branch to the `develop` branch, initiated by user `crisnieto`. The interface shows 0 conversations, 1 commit, 0 checks, and 2 files changed. A comment from `crisnieto` states "First feature for application". Below the comment, a commit titled "First code introduced" is shown with a green checkmark and the commit hash `df8831f`. A message indicates that more commits can be added by pushing to the `feature/first-code` branch on the repository `mercadolibre/fury_release-process-demo`.

The checks section shows the following status:

- Review required** (red X icon): At least 1 approving review is required by reviewers with write access. [Learn more.](#)
- All checks have passed** (green checkmark icon): 4 successful checks. [Hide all checks](#)
- continuous-integration** (green checkmark icon): Jenkins loves your pull request. **Required** [Details](#)
- minimum-coverage** (green checkmark icon): 57.98%. Melicov loves your pull request. **Required** [Details](#)
- pull-request-coverage** (green checkmark icon): 100.00%. Melicov loves your pull request. **Required** [Details](#)
- workflow** (green checkmark icon): Gitflow loves your pull request. **Required**

Merging is blocked (red X icon): Merging can be performed automatically with 1 approving review.

At the bottom, there is a "Merge pull request" button and a note: "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)."

¿Qué ejecutamos en CI?



✓ release-process-demo 1

Pipeline

Changes

Tests

Artifacts



Login



Branch: —

🕒 34s

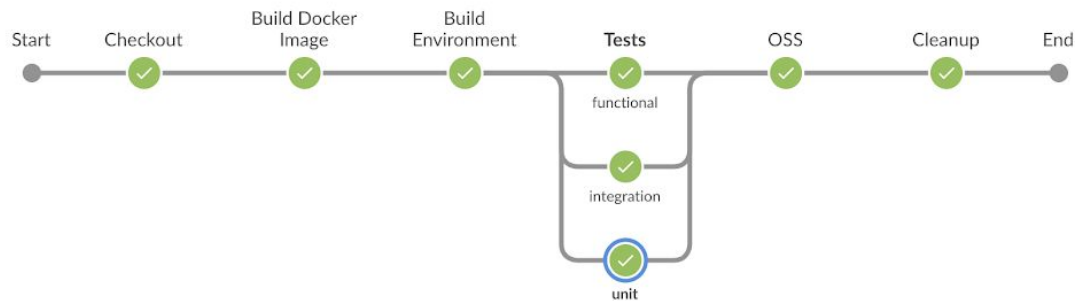
No changes

Commit: —

🕒 4 minutes ago

Started by user Admin

Description PR #1 - https://github.com/mercadolibre/fury_release-process-demo/pull/1



// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

// **CODE COVERAGE**





IT BOARDING

BOOTCAMP

// Code Coverage

Existe una herramienta de Code Coverage, llamada Melicov, desarrollada *in-house* por el equipo de Release Process que nos permite obtener información tanto cuantitativa como cualitativa sobre la cobertura de código de una aplicación.

Características principales:

-  Establecer y mantener un estándar de calidad esperado como equipo.
-  Entender si el código introducido en un nuevo desarrollo está correctamente testeado y detectar falencias.
- ☒ Comparar dinámicamente la cobertura entre el nuevo código (**head branch**) y el original (**base branch**).
-  Visualizar las líneas de código cubiertas y no cubiertas. No sólo de todos los archivos del proyecto, sino también de aquellos que únicamente fueron modificados en un pull request.
-  Entender el porcentaje de cobertura no sólo de cada archivo, sino también de la aplicación entera.

Controles de Code Coverage

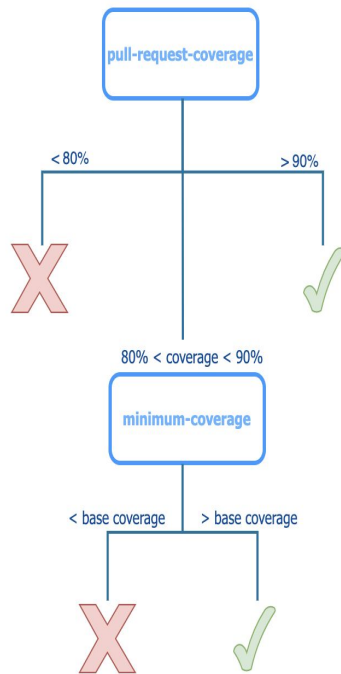
Para garantizar la calidad de los cambios introducidos, luego de ejecutar los tests y garantizar que fueron exitosos, Release Process va realizar el control de la cobertura de código resultante de esos tests.

El check `pull-request-coverage` nos indica si el porcentaje de líneas cubiertas para el total de líneas agregadas/modificadas exclusivamente en el pull request en cuestión. Básicamente significa que se evalúan únicamente los cambios introducidos en el pull request.

El check `minimum-coverage` nos indica si el porcentaje de cobertura de código de la aplicación entera alcanza el mínimo requerido. Es decir, que se consideran todos los archivos de la aplicación.

Melicov toma como prioridad el `pull-request-coverage`, el cual debe ser mayor a 80%. En caso de que lo sea, el check pasará a validar el `minimum-coverage`, el cual se calcula en función del porcentaje de `pull-request-coverage`.

En caso que el `pull-request-coverage` sea mayor o igual a 90%, el check de `minimum-coverage` será aprobado automáticamente. En caso que se encuentre entre 80 y 90%, el nivel de cobertura total deberá ser mayor al último registro que tenga melicov para el *base branch* en cuestión.



✓	🔴	<code>minimum-coverage</code> — 87.91%. Melicov loves your ...	Details
✓	🔴	<code>pull-request-coverage</code> — 100.00%. Melicov loves y...	Details

// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

// OTROS CHECKEOS

IT BOARDING

BOOTCAMP

SAST

- Control automatizado que reporta a través del check de vulnerabilidades cualquier tipo de vulnerabilidad en el código como credenciales hardcodeadas, SSRF (Server side request forgery) y SQLI (SQL Injection)

Dependencies

- Se encarga de validar si existen dependencias externas que presenten vulnerabilidades ya reportadas o descubiertas.

Code Quality

- Es un servicio desarrollado in-house para evaluar la calidad de nuestro código, utilizando análisis estático como una de las principales fuentes de información

// VERSIONES Y SEMVER

IT BOARDING

BOOTCAMP

// Versiones Automáticas

Haciendo un uso correcto de **Gitflow** en Release Process podemos acceder a la funcionalidad de creado automático de versiones que nos permite con el menor esfuerzo posible generar versiones para desplegar en ambientes de test / beta y producción

Cuando creamos un pull request desde `release/*` o `hotfix/*` y hacia `branch master` y todos los controles de Release Process son exitosos, vamos a ver en el pull request que una versión comienza a crearse.

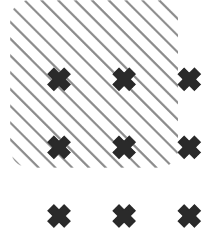
También es posible crear versiones de test en branches `feature/*` escribiendo un comentario en el pull request que diga `rp build test` Esto es útil para poder probar la aplicación en servidores de test.

rp-build-service bot commented 4 minutes ago

Build report

Status: *finished* ✓

Version: 1.1.0-rc



Semver

Todas las versiones que creamos para las aplicaciones de Fury tienen un **Versionado Semántico 2.0.0** (Semver). Es decir que los nombres de las versiones están basados en tres dígitos.

- major: dígito utilizado para indicar cambios mayores. Por ejemplo, cambios de firma, tecnología, etc.
- minor: dígito utilizado para indicar cambios menores. Por ejemplo: una nueva funcionalidad.
- patch: dígito utilizado para indicar cambios rápidos y pequeños. Por ejemplo: un hotfix.

La composición del nombre es entonces: mayor.minor.patch. Por ejemplo: 1.0.0

A su vez, existen Flavors de las versiones, los cuales, al usar Gitflow, son inferidos automáticamente. Los flavors son:

- Release Candidate -> 1.1.0-rc-1
- Hotfix -> 1.0.1-hotfix-1
- Test -> 1.0.1-test-1
- Release -> 1.1.0

// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

// ¿QUÉ ES UN CODE REVIEW?

IT BOARDING

BOOTCAMP



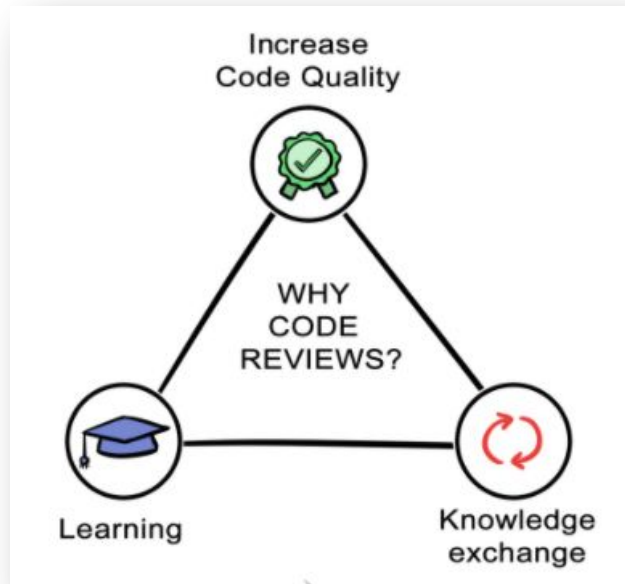
// Code Review

Es el **examen sistemático del código fuente por parte de los miembros de un equipo**. Se realiza con el objetivo de **mejorar la calidad** del código mediante la detección temprana de errores o alternativas más eficientes a la implementación inicial.

Sirve para mejorar las cualidades de los desarrolladores, inculcar buenas prácticas y discutir diferentes aproximaciones a la solución.

Beneficios:

- **Compromiso y motivación** de los desarrolladores para escribir un código más limpio.
- **Comunicación y conocimiento compartido:** Sirve para que el equipo se comunique y esté alineado.
- **Aprendizaje:** Ayuda a los nuevos desarrolladores a integrarse y adoptar las prácticas del equipo al que se unen, y obtener feedback sobre su trabajo.
- **Consistencia:** El código es propiedad y responsabilidad de todo el equipo, y debería ser uniforme. Es importante establecer reglas de estilo, estructura de los proyectos, etc.



// ¿CÓMO HACER UN CODE REVIEW?

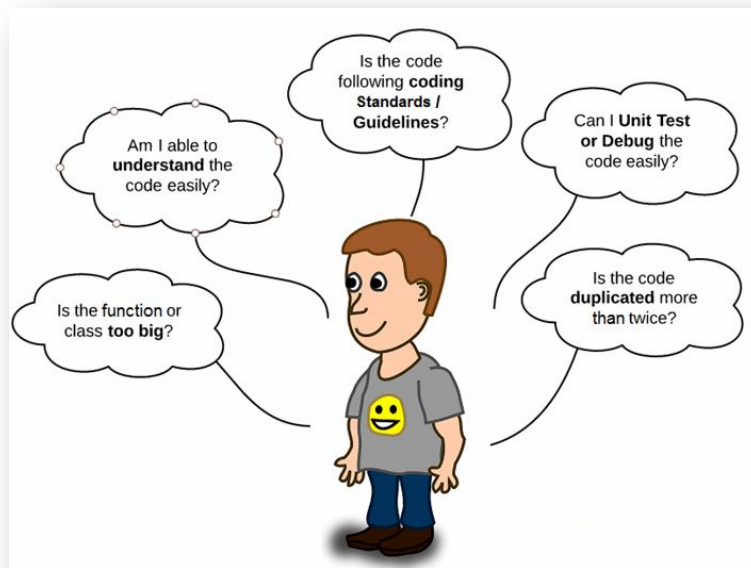
IT BOARDING

BOOTCAMP

// ¿Cómo hacer un Code Review?

Todos pueden aportar algo en el Code Review. **Para hacerlo deberíamos tener en cuenta algunos puntos como:**

- **Propósito:** Cada cambio debe tener una razón. ¿El código cumple con los requerimientos y soluciona el problema planteado?
- **Legibilidad y estilo:** Nombres de las variables y los métodos, expresividad y elegancia. Ej.: qué tan coherente es el verbo http con la acción que realiza. Respeto de estilo y convenciones. ¿Se podría escribir diferente? Ej.: reemplazar un `forEach()` por una expresión lambda.
- **Mantenibilidad:** Dependencias entre clases. Clases o métodos muy grandes.
- **Seguridad:** Validaciones (inputs), cifrado.
- **Arquitectura Software:** ¿El código construido es flexible, escalable y reutilizable? ¿Utiliza los patrones de diseño?.
- **Rendimiento:** ¿Se utiliza la cache de manera correcta?.
- **Calidad:** ¿Los test unitarios prueban casos corner y edge?.



NO se trata de una crítica al autor, sino de una instancia de aprendizaje, reflexión y crecimiento para el desarrollador y todo el equipo.

// HERRAMIENTAS PARA HACER UN CODE REVIEW

IT BOARDING

BOOTCAMP

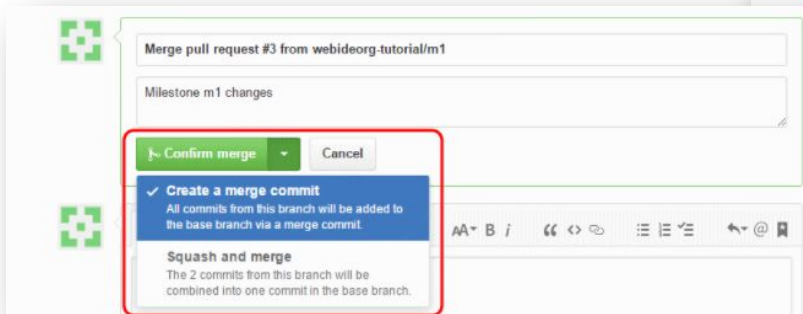
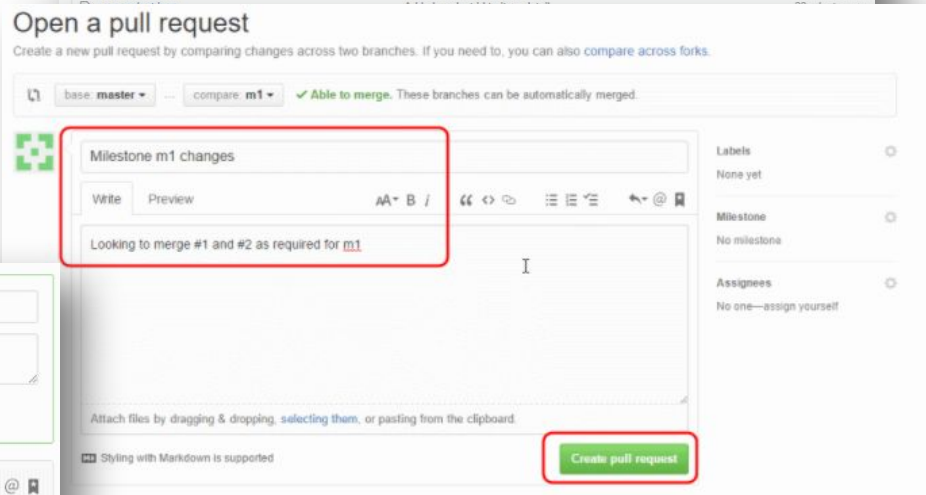
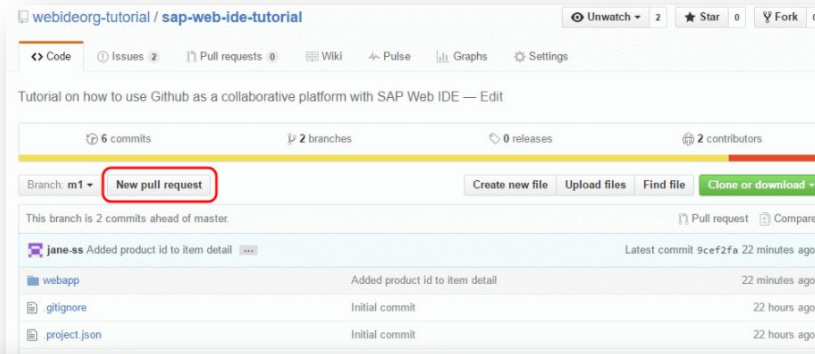
// Github

Para hacer un CR es necesario **crear una branch** en el repositorio de proyecto que tenemos en github.com.

Una vez que nuestro nuevo código se encuentre listo para ser revisado vamos a **crear un Pull Request**.

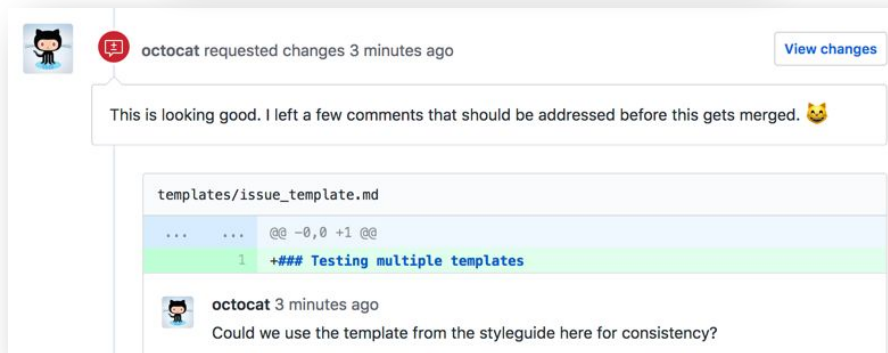
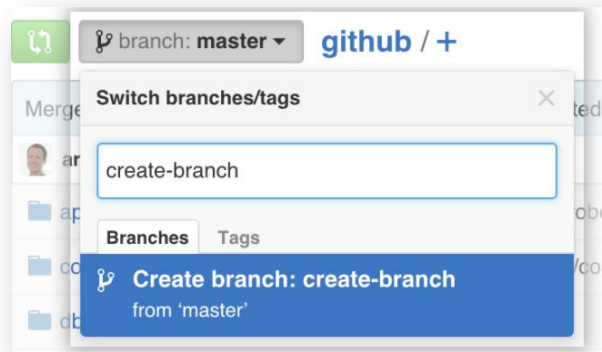
En esta instancia nuestros compañeros de equipo realizarán el Code Review, pudiendo dejar comentarios sobre nuestro código.

Una vez que todos los comentarios hayan sido resueltos, y el **PR aprobado** por los reviewers, **la branch con nuestro nuevo código podrá ser «mergeada»** es decir, integrarse al master del repositorio.



¿Dónde se hace un Code Review?

El **Code Review** se hace sobre los **Pull Requests**. Los miembros del equipo realizan comentarios sobre las líneas de código, proponiendo cambios o realizando preguntas que incentiven la discusión sobre la solución planteada.



// **iHANDS ON TIME!**

IT BOARDING

BOOTCAMP

IT BOOTCAMP | **QUALITY**

// **DEMO**

IT BOARDING

BOOTCAMP

IT BOOTCAMP | **QUALITY**

// **iGAME TIME!**

IT BOARDING

BOOTCAMP



Gracias.

IT BOARDING

BOOTCAMP

