.UBAfiuba
FACULTAD DE INGENIERÍA

Taller de Programación I

Flujos de desarrollo en GIT

Ing. Pablo A. Deymonnaz - Uriel Kelman
pdeymon@fi.uba.ar - urielkelman@gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

.UBAfiuba FACULTAD DE INGENIERÍA

Índice

¿Qué es GIT?

- Git es un sistema de control de versiones distribuido gratuito, open source (código abierto), diseñado para el manejo de proyectos de diversos tamaños (pueden ser grandes o pequeños) con rapidez y eficiencia.
- Fue desarrollado en 2005 por Linus Torvalds (quien a su vez es el creador del kernel del sistema operativo Linux) y actualmente sigue en mantenimiento continuo (la versión más nueva es la 2.26.2).
- El principal uso de Git como herramienta busca lidiar con la necesidad de poder tener versionado el código de un proyecto.

¿Qué significa **control de versiones?** (1)

- Cuando trabajamos sobre un proyecto, hay un flujo que se repite constantemente: creamos archivos nuevos, los editamos, los guardamos, los volvemos a abrir, los volvemos a editar, los volvemos a guardar y así sucesivamente.
- El control de versiones nos permite dejar un registro del momento en el que guardamos el código de un proyecto, guardando en dicho registro cuándo hicimos el cambio, por qué lo hicimos, y qué fueron las cosas que cambiamos respecto a la versión.
- El control de versiones nos permitirá tener una **historia** del código del proyecto.

¿Qué significa **control de versiones?** (2)

- Para un único individuo trabajando en el proyecto, podríamos pensar que mantener un registro de la historia del mismo es relativamente sencillo: bastan un par de anotaciones del mismo y ya.
- Aquí es donde Git saca a relucir su brillo: cuando trabajamos en un proyecto en forma **colaborativa**. No es lo mismo trabajar sólo en un proyecto que trabajar con un grupo de personas las cuáles quieren modificar el código del proyecto al mismo tiempo (incluso el código de los mismos archivos).
- Git, como sistema de control de versiones, nos brindará todas las herramientas necesarias para poder trabajar de manera controlada de forma colaborativa en un proyecto junto a otros programadores.

Github

- Github es una plataforma que permite el almacenamiento de proyectos Git
- ¿Es la única que existe? No, existen algunas otras que también son muy utilizadas como por ejemplo BitBucket o Gitlab.
- Github (como cualquier otra plataforma de almacenamiento) agregan la abstracción del repositorio: un lugar donde todos los archivos pertenecientes a un proyecto pueden ser guardados. Cada proyecto tiene un único repositorio, identificado con una única URL.
- El 4 de junio de 2018, Microsoft compró GitHub por el total de 7500 millones de dólares.

Operaciones habituales en GIT

- Crear/clonar un repositorio.
- Crear un rama (*branch*).
- Agregar los cambios realizados sobre un *branch*.
- Commitear los cambios realizados.
- *Pushear* los cambios al repositorio remoto.

.UBAfiuba FACULTAD DE INGENIERÍA

Índice

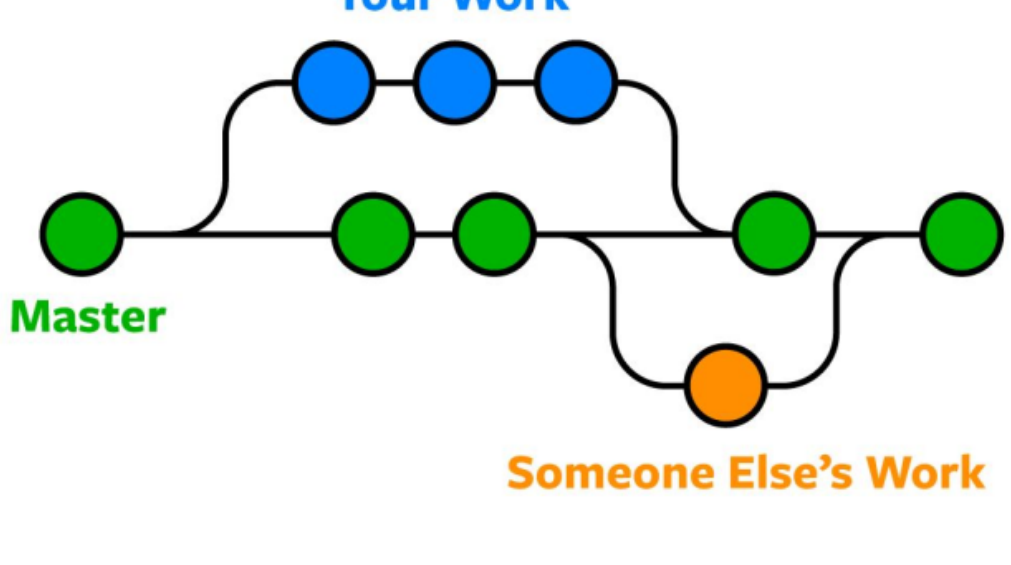
Ramas (1)

Podemos pensar a una rama como una unidad para escribir código aislada e independiente. Esto tiene una serie de ventajas:

- Cambios de contexto sencillos.
- Ramas con distintos roles (*production*, *testing*, etc.)
- *Workflow* basado en features.
- Experimentación desechable.

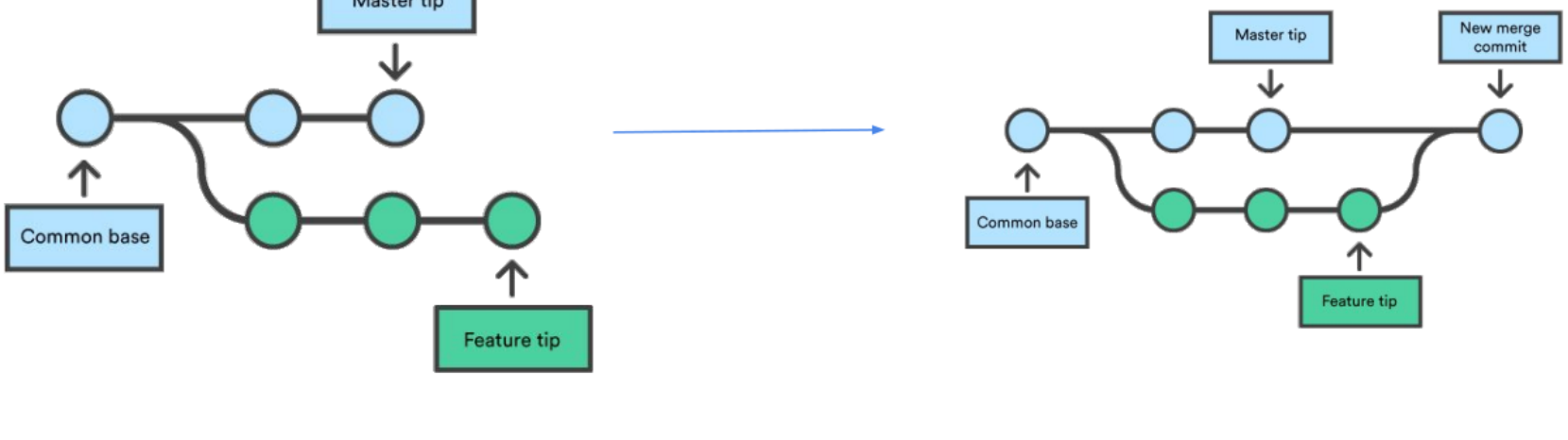
Ramas (2)

Resulta de vital interés poder trabajar en distintas ramas al mismo tiempo. De esta forma, distintos desarrolladores pueden trabajar en distintas funcionalidades en forma aislada.



Mezclando ramas

Una vez que terminamos de trabajar en una rama, lo habitual es querer que dicha funcionalidad quede mezclada con todas las anteriores. Para esto, debemos realizar un *merge* entre las ramas que queremos mezclar.

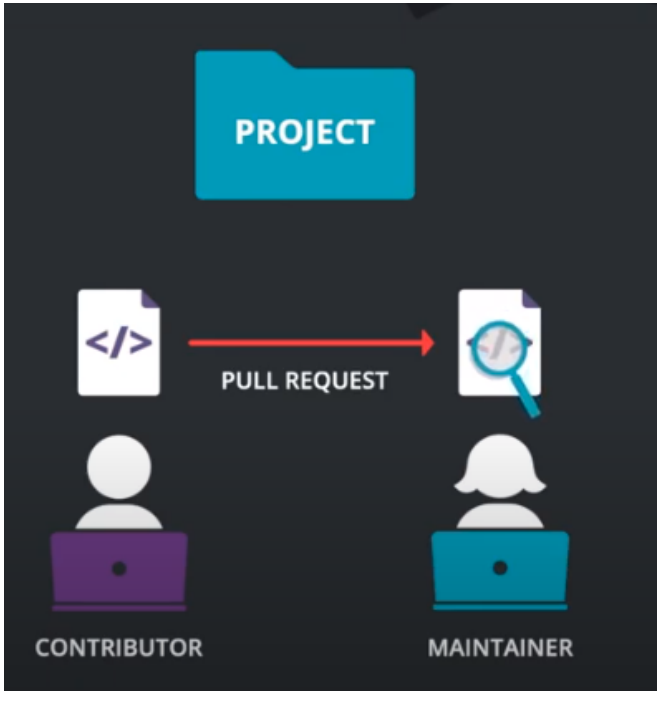


.UBAfiuba FACULTAD DE INGENIERÍA

Índice

Definiendo un PR

Un *pull request* (también conocido como *merge request*) es un evento en donde un desarrollador solicita una revisión sobre código que implementa una nueva funcionalidad y que desea *mergear* a alguna rama del proyecto (generalmente la rama principal).

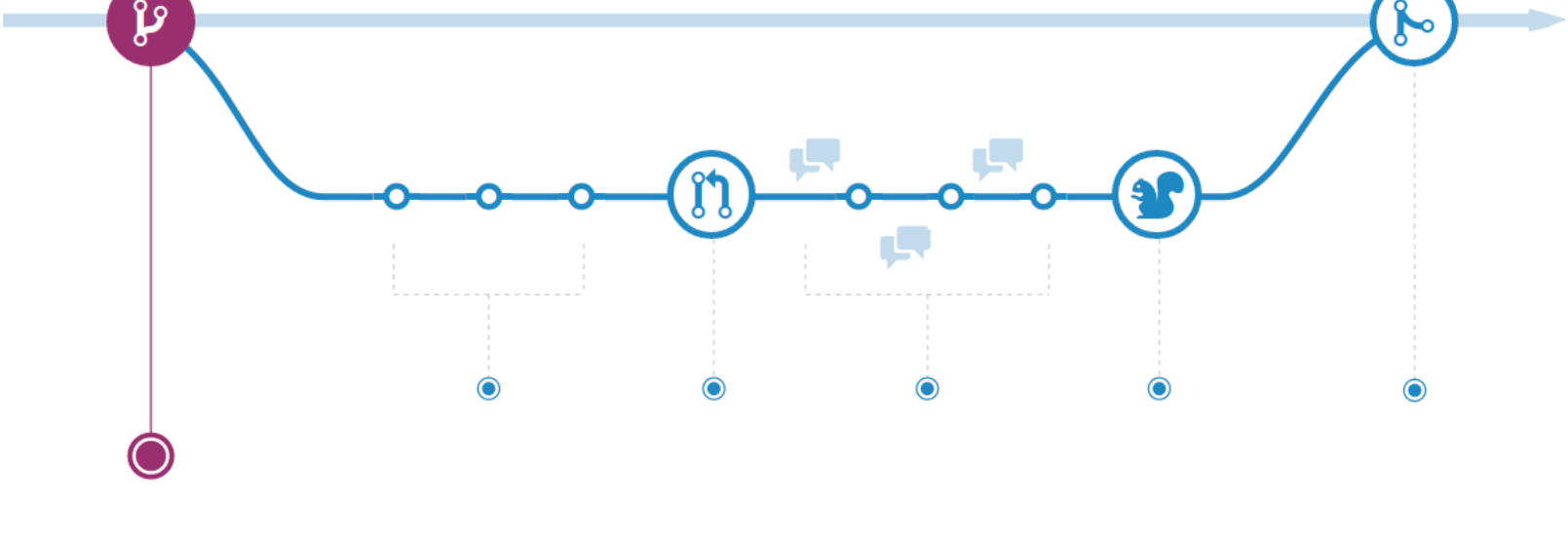


Ventajas de trabajar con PRs

- Obliga a realizar revisiones de código antes de mezclar código nuevo a alguna rama.
- Permite que se debatan cambios sobre una funcionalidad antes de llevarla a un ambiente productivo.
- Proveen una forma natural de dar *feedback* sobre el código que escribió algún colaborador.

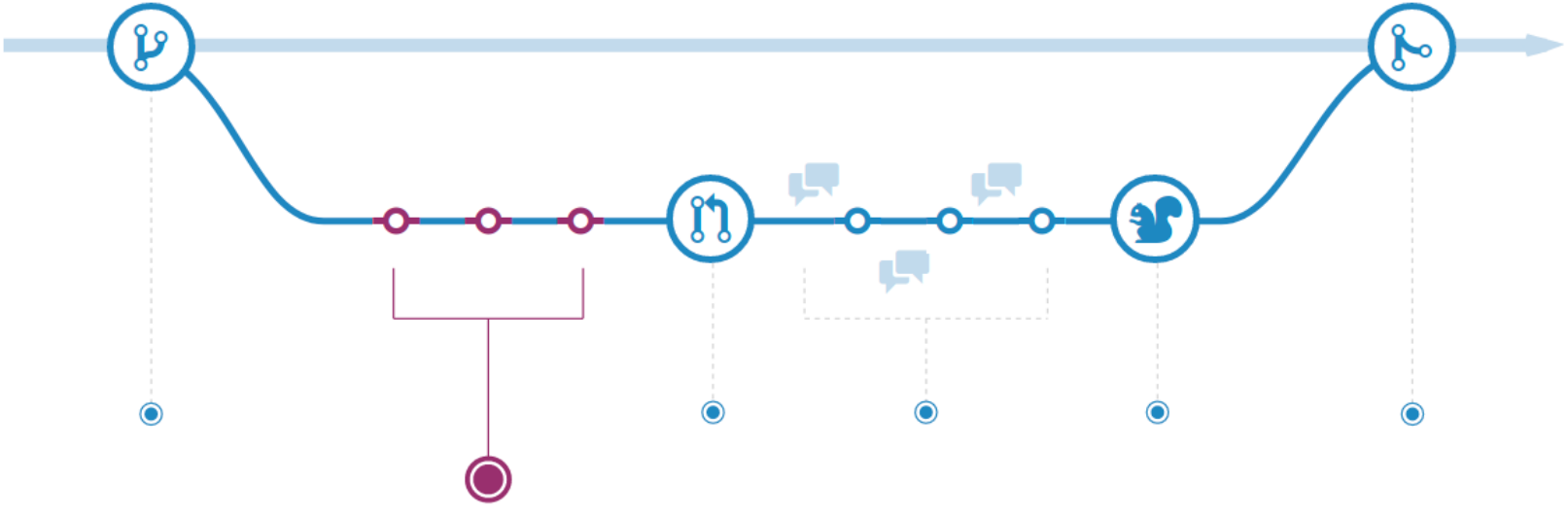
Github Flow (1)

Crear un *branch* en base a un *feature* que queremos implementar.



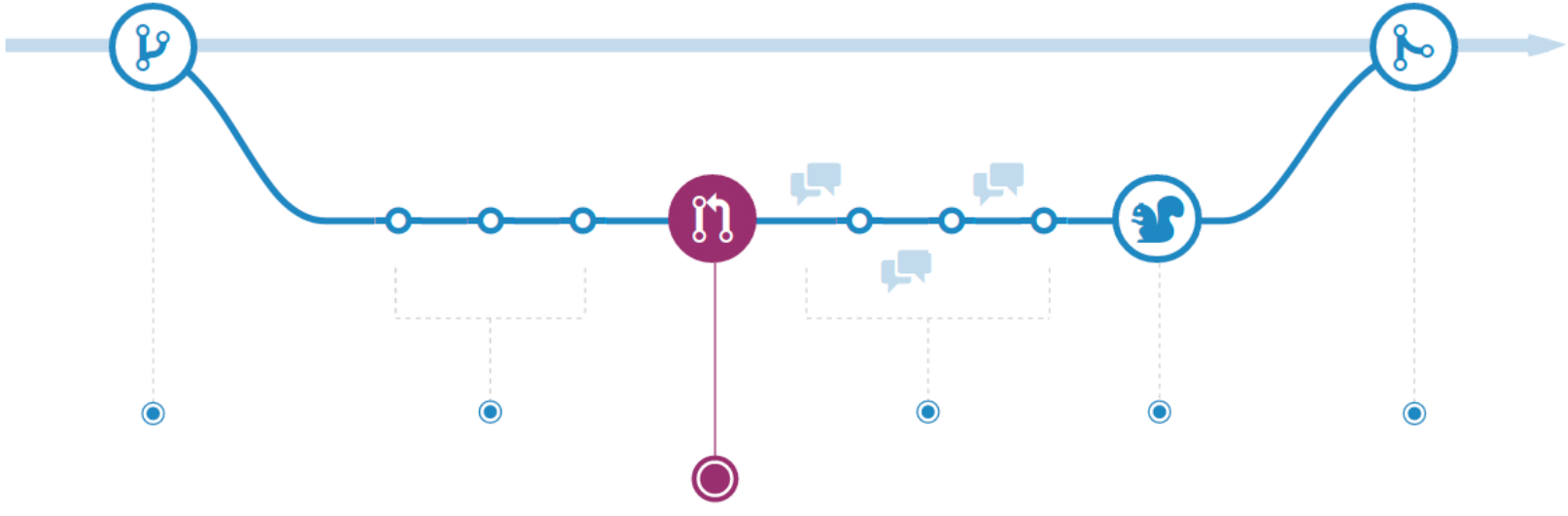
Github Flow (2)

Agregar cambios y *commits* sobre la rama en la que se está trabajando.



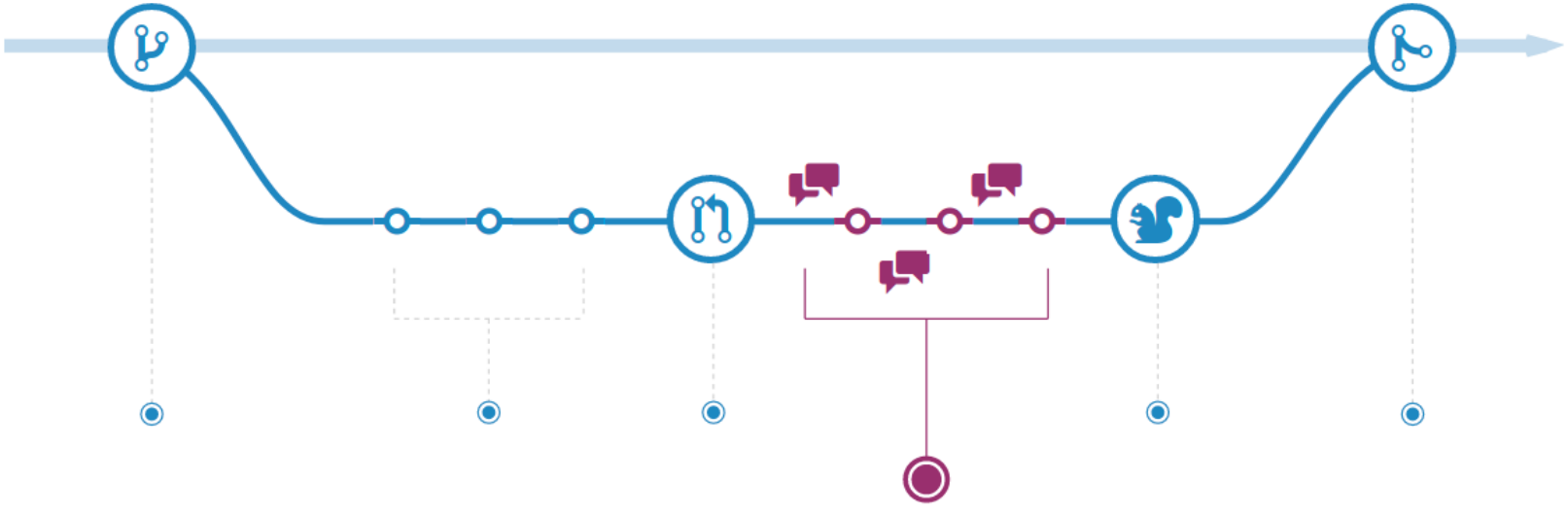
Github Flow (3)

Una vez que terminamos el *feature*, abrimos un *pull request* que apunta a la rama contra la que queremos realizar el *merge*.



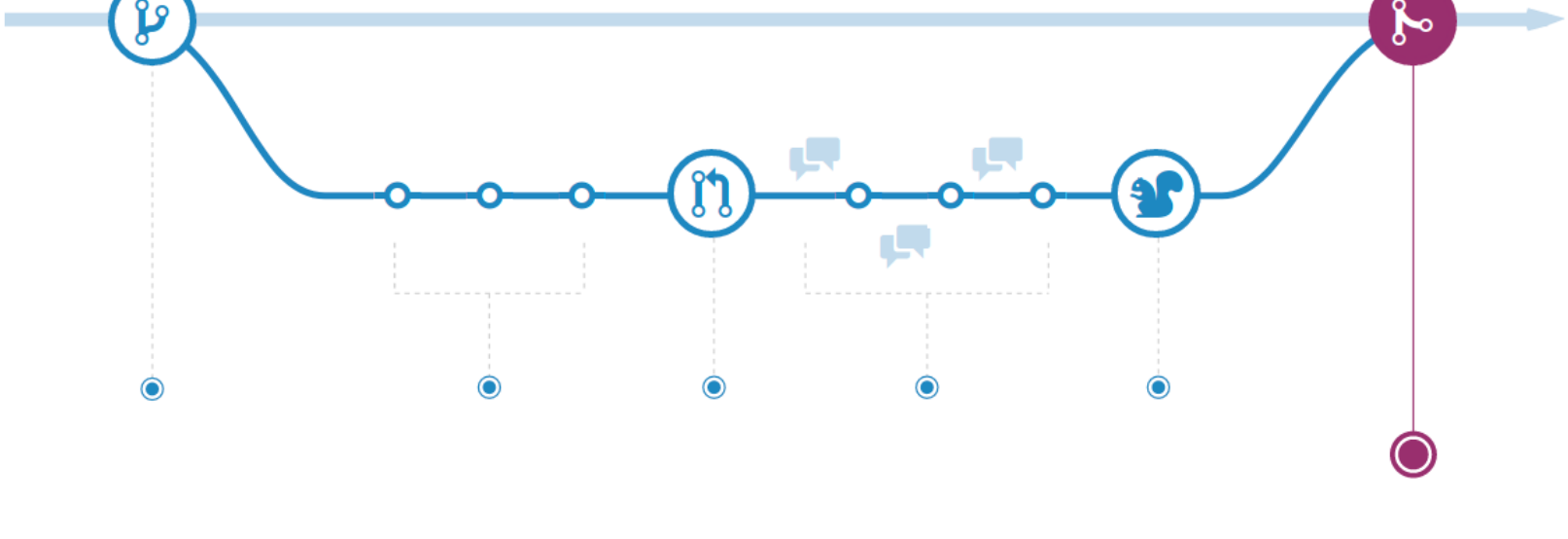
Github Flow (4)

Revisión del código y posible discusión acerca del mismo.



Github Flow (5)

Aprobación del *pull request* y merge.



Síntesis

1. El desarrollador terminar el *feature* y abre un *pull request* que apunte a la rama correspondiente.
2. El PR está abierto para que otro desarrollador lo revise e inicie una discusión.
3. Quien revise el PR, otorga *feedback* en forma de comentarios.
4. El desarrollador responde a los comentarios: a veces realizando modificaciones a su códigos, otras veces contestando dudas o explicando alguna decisión de desarrollo.
5. Una vez que todos los comentarios son resueltos, se puede aprobar el PR y realizar el *merge*.

Conflictos

- Al fusionar dos ramas, o al intentar realizar un pull request, es posible que git nos avise que existe un conflicto entre las ramas que se quiere mergear.
- Esto sucede cuando existen modificaciones en ambas ramas para la misma porción de código. En este caso, git no sabe cuál de ambas versiones debe prevalecer, por lo tanto se genera un conflicto.
- Para poder realizar correctamente el merge, debe resolverse el conflicto en forma local, eliminando así las ambigüedades e indicando qué porción de código aparecerá en el repositorio remoto.

Necesidades más complejas

Cuando nos encontramos trabajando en alguna compañía, las necesidades alrededor de un proyecto difieren sustancialmente de las que tenemos en un proyecto académico. Algunas podrían ser:

- Posibilidad de tener más de un ambiente para probar la aplicación.
- Utilizar métodos de CI/CD sobre distintos *branches*.
- Tener algún método organizado de realizar un *hotfix* en caso de que exista algún error en una versión subida a producción.

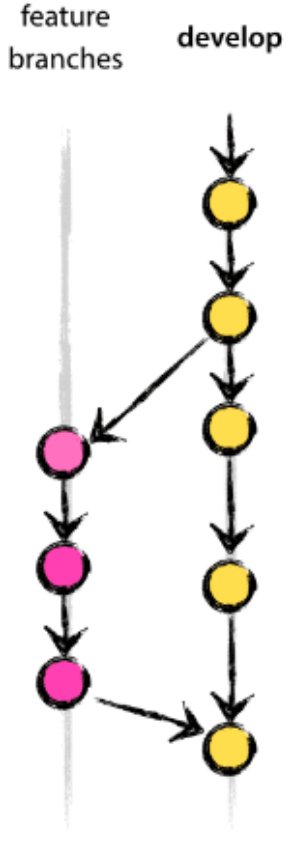
Branches y *workflow* (1)

Un *approach* un poco más sofisticado consiste en tener dos ramas principales en vez de dos:

- **Main:** Contiene la última versión subida a producción.
- **Develop:** De aquí parten los desarrolladores cuando quieren desarrollar un feature. También, realizan el *merge* de los branches de *features* contra este branch.

Branches y workflow (2)

Cuando los desarrolladores desean codificar un *feature*, salen desde el branch develop. Cuando terminan, realizan un PR contra develop



y esperan su aprobación.

Branches y workflow (3)

Una vez que se quiere desplegar una versión en producción porque se considera que se codificaron los suficientes *features*, se crea un branch de *release*. Sobre este branch:

- Se intentan encontrar errores.
- Se realizan *fixes* sobre errores que se pueden encontrar.
- Se puede correr un conjunto de tests de regresión que testean la aplicación integralmente.

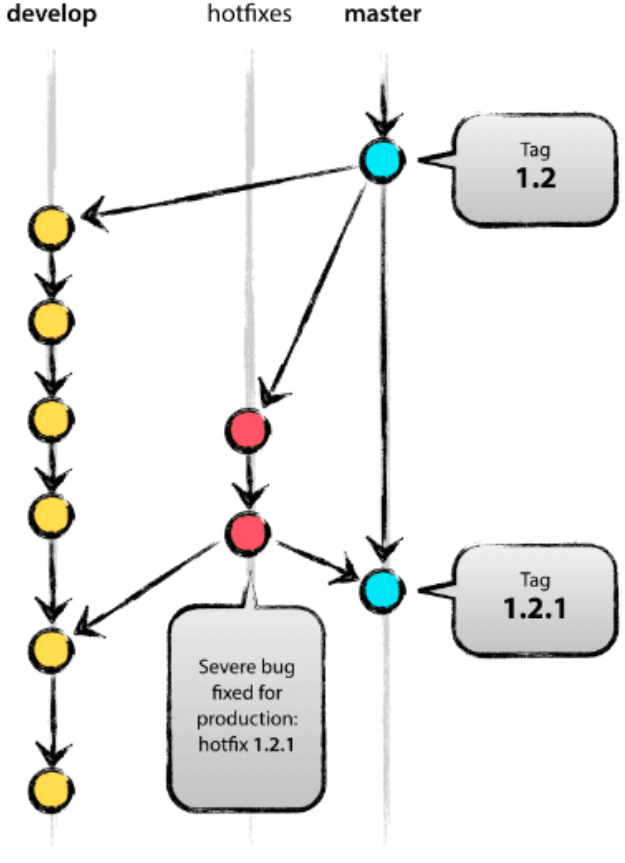
Branches y *workflow* (4)

Una vez que está todo listo:

- Se realiza el merge del branch release tanto contra master como contra develop.
- Se crea un tag en el branch master (debe reflejar el nombre de la versión).
- Se despliega la aplicación en el ambiente productivo.

Branches y workflow (5)

Adicionalmente, pueden crearse branches para realizar *hotfixes*: errores urgentes que no permiten esperar el despliegue de la siguiente versión.



Bibliografía

- Pro Git Book: *https://git-scm.com/book/en/v2*
- Videos introductorios: *https://git-scm.com/videos*
- *https://www.atlassian.com/es/git/tutorials*
- *https://guides.github.com/*
- *https://medium.com/@patrickporto/4-branching-workflows-for-git-30d0aaee7bf*
- *https://blog.axosoft.com/pull-requests-gitflow/*
- *https://nvie.com/posts/a-successful-git-branching-model/*
- *https://open.spotify.com/episode/0QHoMlwANPAp5XYFbiOxCK?si=*