BRANCHING PATTERNS

SCV: Source Code Versioning

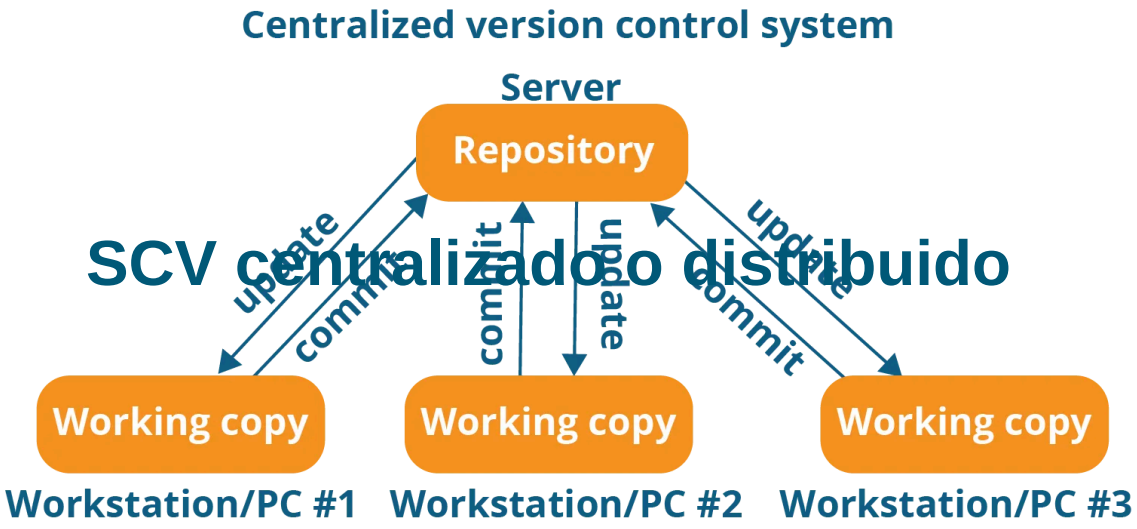
VCS: Version Control Systems

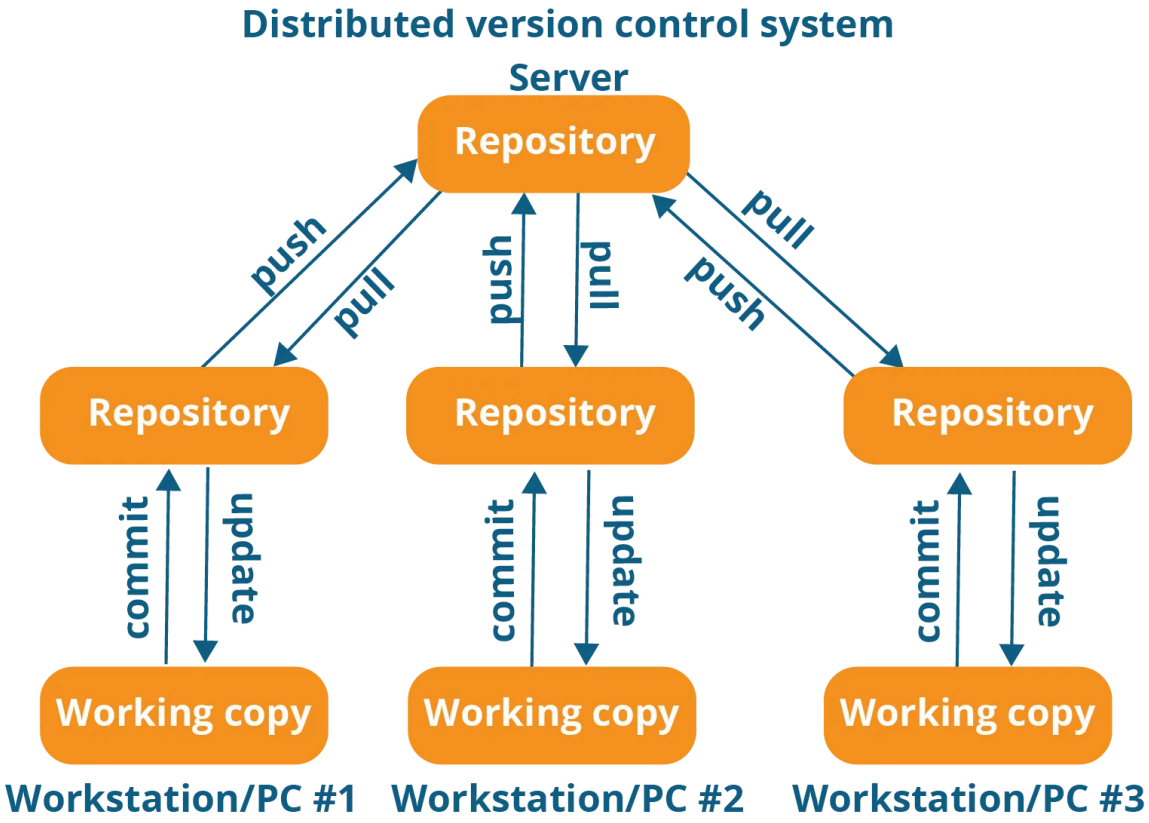
SCM: Source Code Management

¿Para qué sirven?

- Rastrear cambios y restaurar versiones anteriores del software
- Gestionar y coordinar el código fuente en equipos de desarrollo de software
- Seguimiento de varias líneas de trabajo y ayudar a fusionar líneas

SCV centralizado o distribuido





¿Git es descentralidado o centralizado?

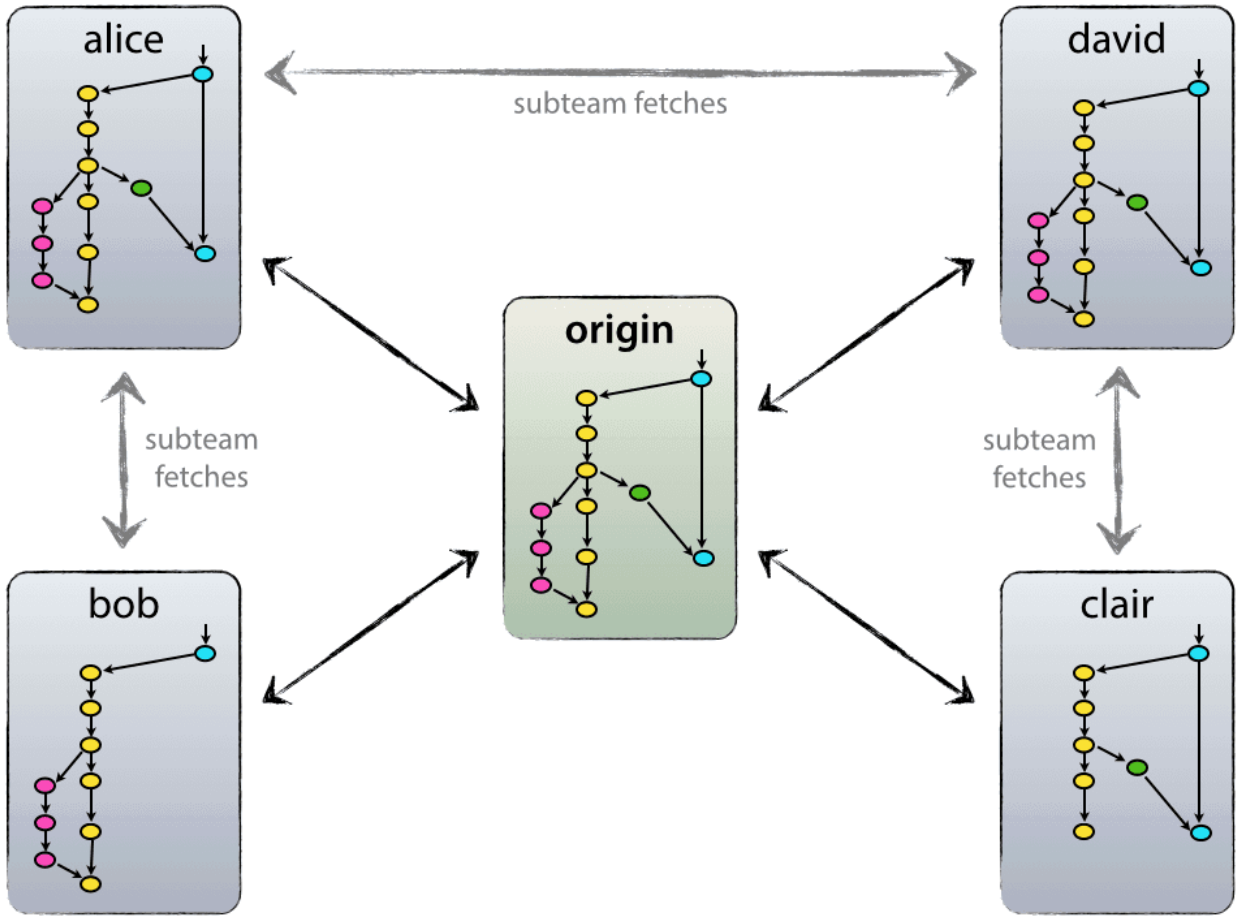
¿Git descentralizado?

Repo central con la verdad

- origin centraliza pull/push

¿Hacer pull de otros repos?

- Subequipos: Alice & Bob, Alice & David, David & Clair
- Alice define *remotes* `bob` y `david`, etc.

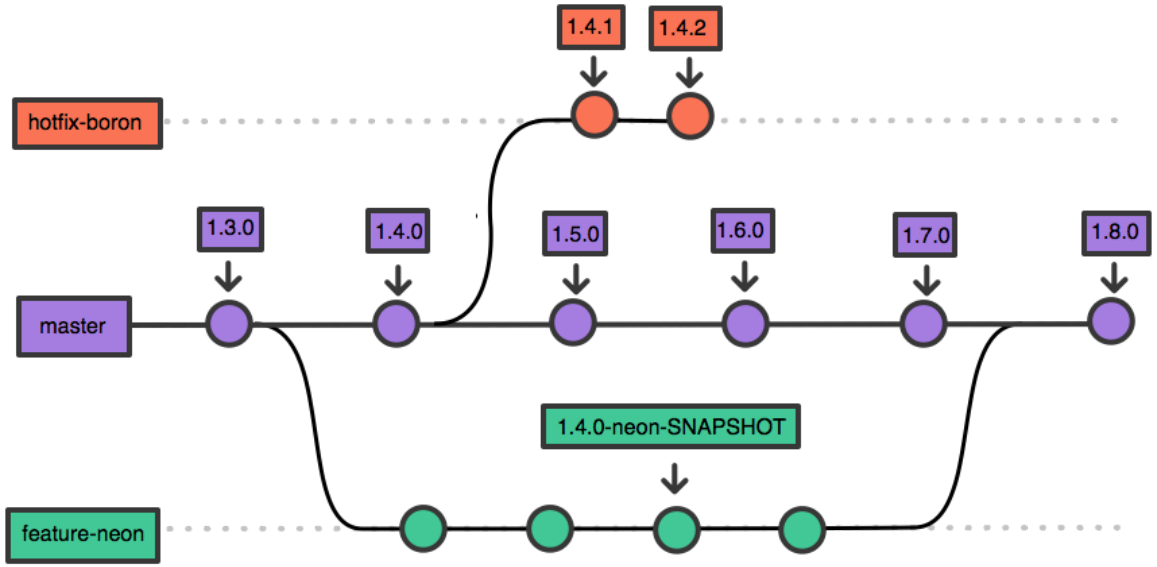


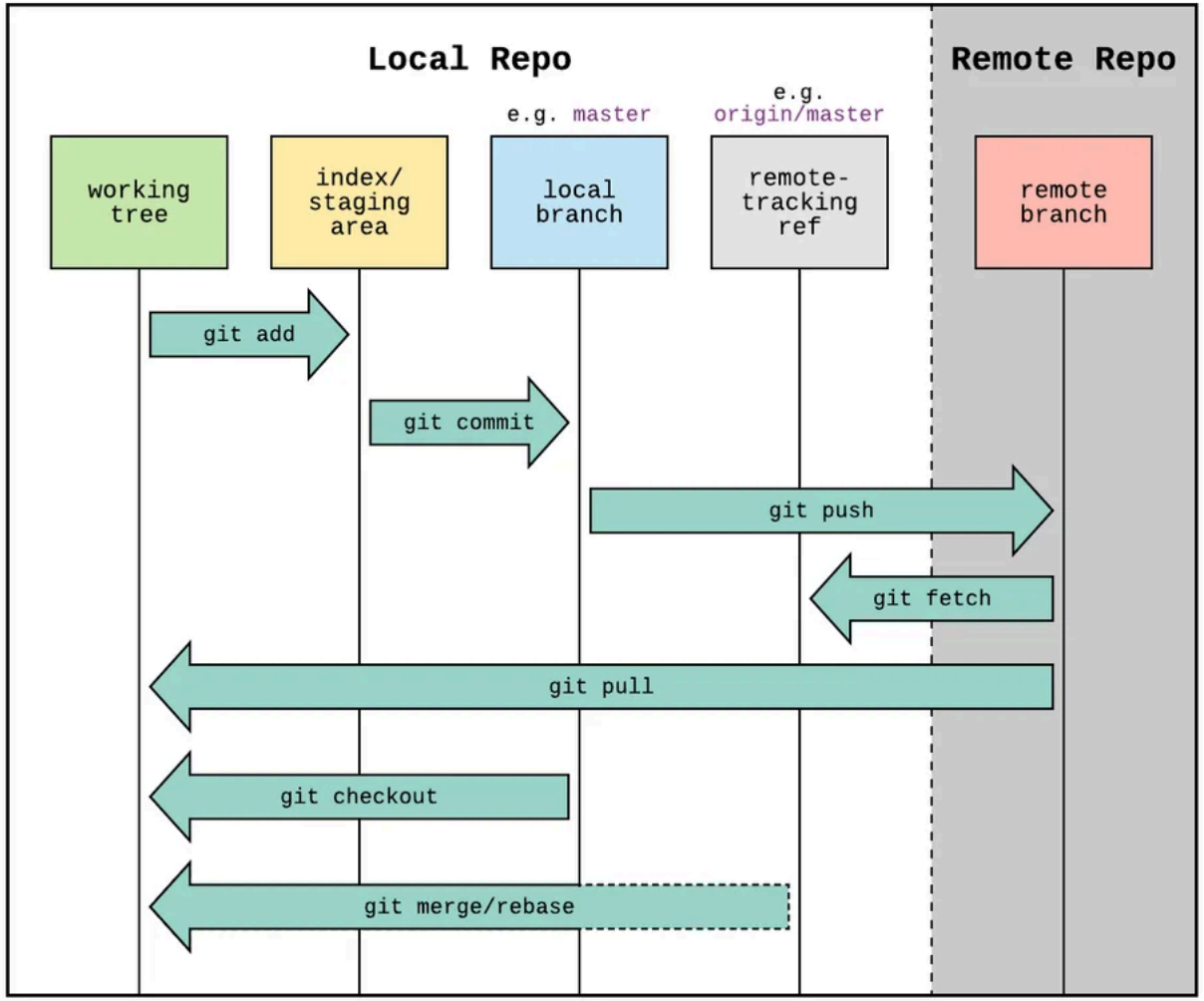
Git: repositorios remotos

	Definiciones
fork	Una copia de un repositorio en el que se puede trabajar de forma independiente (te haces propietario)
clone	Copia de un repo remoto (origin) en la máquina local, para trabajar independientemente (pero no eres propietario)
origin	Nombre para referirse al repositorio remoto del que se clonó un repo local
upstream	Nombre para referirse al repositorio original desde el que se forkeó un repo

Workflow y branching patterns

- Patrones para ayudar a manejar la división del desarrollo en líneas de trabajo que se dividen y fusionan (split/merge) en ramas (branch)
- No son estándares definitivos
- Dependen de la estructura social del equipo y sus prácticas habituales





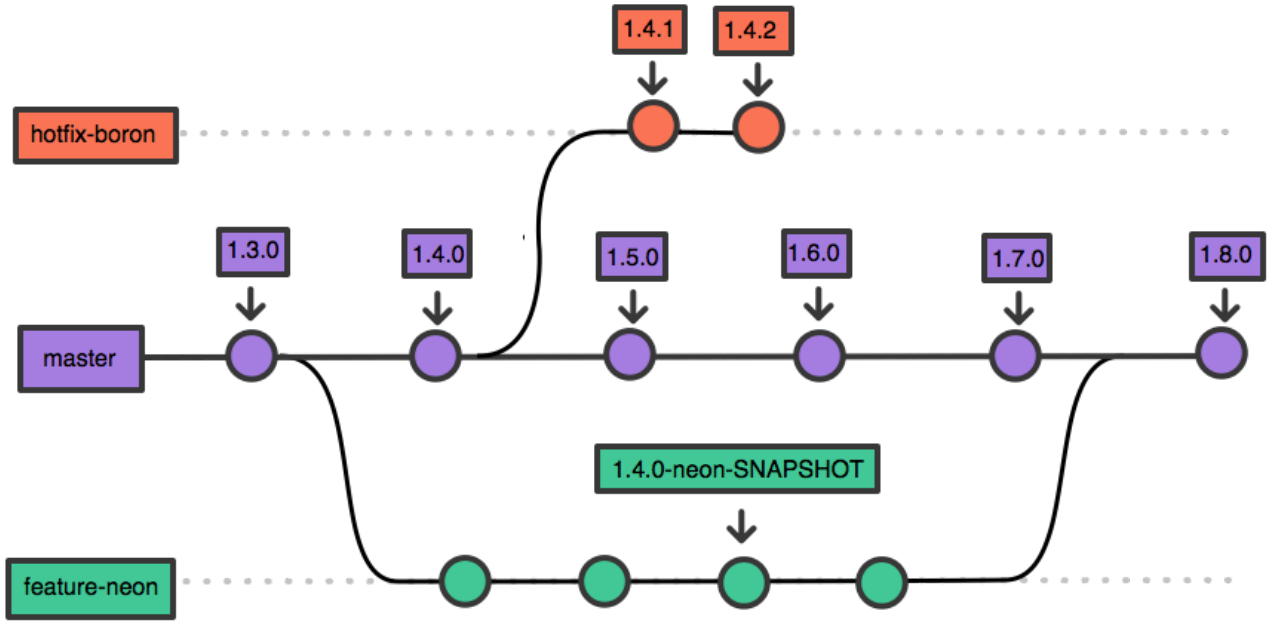
Patrones de workflow habituales

- GitHub flow: para todo (no sólo desarrollo)
- Git Flow: para desarrollo [web], centrado en *releases*
  - Git Flow considered harmful
- OneFlow: ramas de vida corta
- Gitlab flow: menos *tagging* y *merging*

Conceptos de branching

source branching = crear una copia del código fuente para registrar en ella todos los cambios de forma independiente

- ¿Qué es un commit?
- ¿Qué es staging?
- ¿Qué es una rama?
- ¿Qué es un head?
- ¿Qué es un fork?



	Definiciones
commit	Conjunto de cambios en el código fuente que se registra en el repositorio
branch	Una secuencia de commits
head	Último commit de una rama
staging	Preparar los cambios para el próximo commit

¡Cuidado! Confusión terminológica sobre qué es una rama

- ¿Cuando se clona un repo, se crea una nueva rama?
- Distintos DVCS: Mercurial *branch* $\neq$ Git *branch* $\approx$ Mercurial *bookmark*

📙	**Definiciones**
branching	Crear una copia del código fuente para registrar en ella todos los cambios en el código de forma independiente
merge	Integrar los cambios de una rama en otra
codeline	Secuencia particular de versiones de una base de código (*codebase*)
clone	Clonar un repo es hacer un *checkout* de una rama en una nueva *codeline*
mainline	Una rama única compartida por todos que sirve como estado actual del producto

En cada commit, pasar pruebas automáticas para asegurar que la rama no tiene defectos

Codeline

- Cada desarrollador tiene (en cuanto hace cambios locales) al menos una codeline personal en la *working copy* de su equipo local
- En un DVCS como git...
  - ...cuando se clona un repo git, se hace un checkout de main y se actualiza algo, se obtiene una codeline nueva, aunque no se hagan commits
  - ...obtenemos ramas adicionales cada vez que clonamos un repositorio; las ramas pueden ser locales o remotas

Merge vs rebase vs cherry-pick

Merging a branch

Rebasing a branch

Cherry picking

Tutorial recomendado

JJ Merelo: Aprende Git

Patrones habituales de Workflow

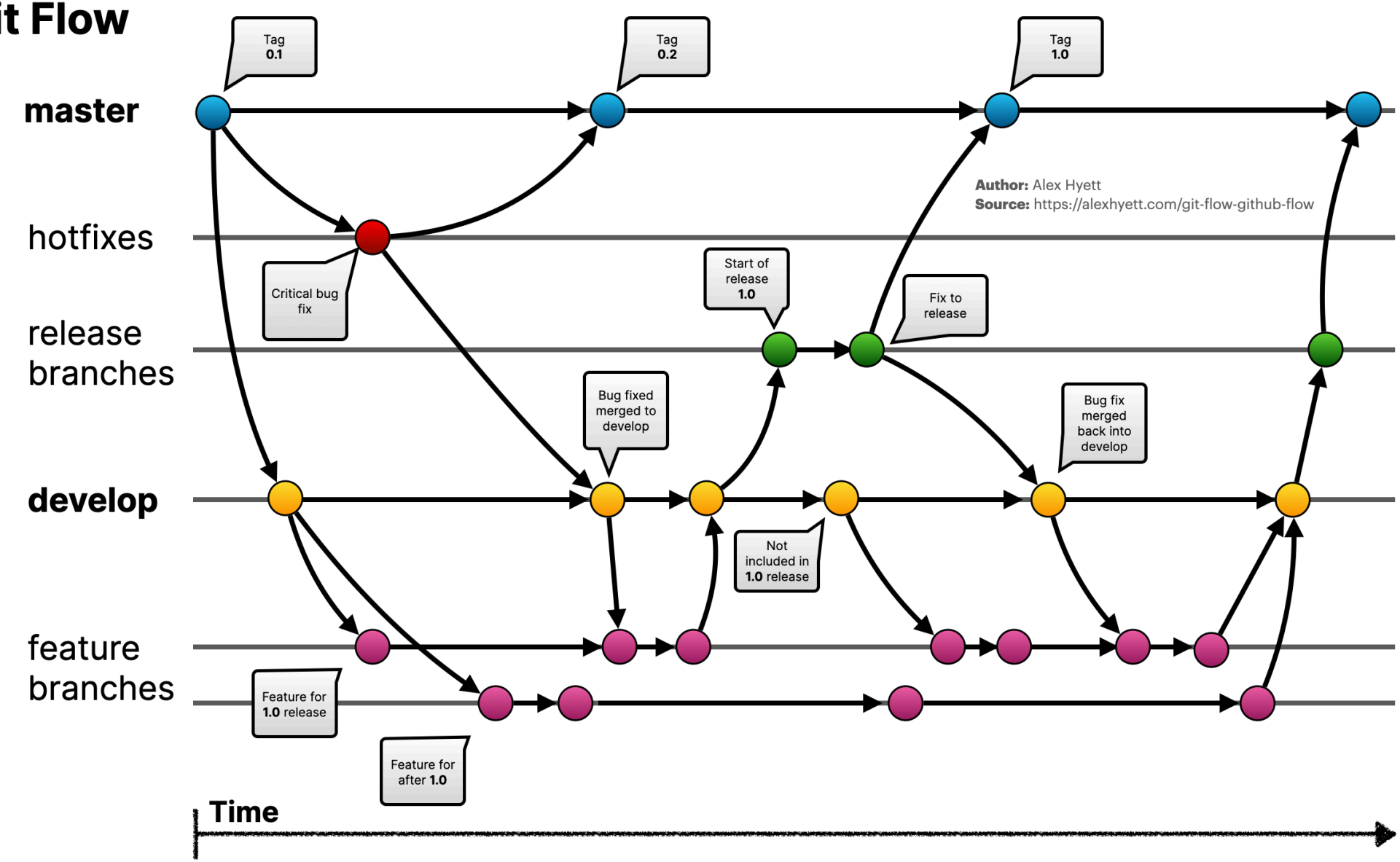
1. Git Flow

Más estructurado y utiliza diferentes tipos de ramas para diversas etapas del desarrollo.
No es muy adecuado para hacer CD o CDEP.

1. La rama `develop` es la base para nuevas características.
2. Se crean ramas `feature` de características y se fusionan de nuevo en `develop`.
3. Se crean ramas `release` para preparar las versiones a entregar.
4. Una vez probadas, las ramas se fusionan en `master` y `develop`.
5. Las ramas `hotfix` corrigen problemas en producción y se fusionan en `master` y `develop`.
6. Cuando los cambios en `develop` son estables, se fusionan en `master` y se le asigna una *tag* con el número de *release*.

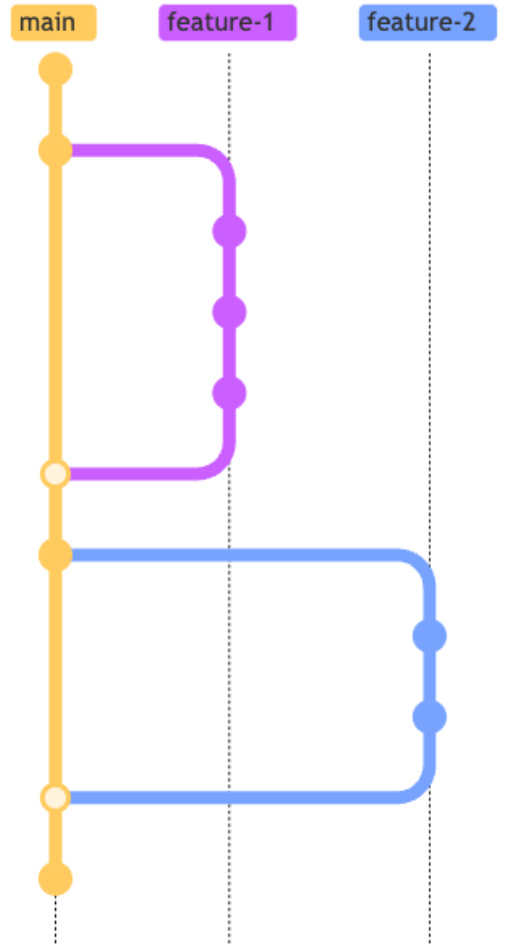
Alex Hyett: Comparativa GitHub flow vs GitFlow

Git Flow


Tag
0.1
Tag
0.2
Tag
1.0
master
hotfixes
release
branches
develop
feature
branches
Author: Alex Hyett
Source: https://alexhyett.com/git-flow-github-flow
Critical bug
fix
Start of
release
1.0
Fix to
release
Bug fixed
merged to
develop
Bug fix
merged
back into
develop
Not
included in
1.0 release
Feature for
1.0 release
Feature for
after 1.0
Time

2. GitHub Flow

Enfoque sencillo para CD/CDEP. No para características complejas con muchos desarrolladores en paralelo. Despliegues deben estar automatizados.



1. Una rama `main` o `master` como rama principal.
2. Se trabaja en ramas aparte para nuevas características.
3. Las correcciones rápidas se tratan como características.
4. Se crean *pull requests* para discutir y revisar los cambios.
5. Una vez aprobados, los cambios se fusionan en la rama `main`.
6. Todo en la rama `main` está listo para ser desplegado.

Scott Chacon: GitHub Flow

Git Flow vs GitHub Flow

	Git Flow	GitHub Flow
Adaptabilidad	Puede ser excesivo para proyectos pequeños	Más simple y adaptable para proyectos ágiles
Releases	Varias versiones en producción	Una versión en producción
Colaboración	Énfasis en separación de roles	Colaboración más sencilla y continua
Despliegue	Potencialmente más lento (muchas ramas)	Más rápido (CI se hace en main)
Automatización	Se beneficia de herramientas específicas	Requiere despliegue automático

3. GitLab Flow

Versión simplificada y más orientada a la entrega continua

1. Se trabaja con ramas `feature` y `fix` para nuevas características y correcciones de errores.
2. Se crean *merge requests* para revisar y discutir los cambios.
3. Una vez aprobados, se fusionan en `main`.
4. Se crean ramas `production` y `stable` para producción.
5. Define un conjunto de buenas prácticas.

NOTA: Los merge requests se llaman *pull requests* en git

Ejercicio recomendado

Tutorial de `git-flow`

Tipos de patrones

Martin Fowler: Patterns for managing source code branching

- Patrones de integración: cómo combinar el trabajo de varios desarrolladores
- Patrones de entrega: cómo llevar una *codebase* a producción
- Otros patrones de base
  - Source branching
  - Mainline
  - Healthy branch

Patrones de integración

Patrones de integración

- Mainline integration: Los desarrolladores integran su trabajo haciendo *pull* de la mainline, fusionando y haciendo *push* a la mainline que debe quedar saludable.
- Feature branching: Una rama propia para todo el trabajo de cada característica e integrarla en la mainline al completar la característica.
- Continuous integration: Los desarrolladores integran su trabajo en la mainline tan pronto como tienen un commit saludable que compartir (normalmente < 1 día).
- Pre-integration review: pull/merge requests

Martin Fowler: *Patterns for managing source code branching*

Integration patterns

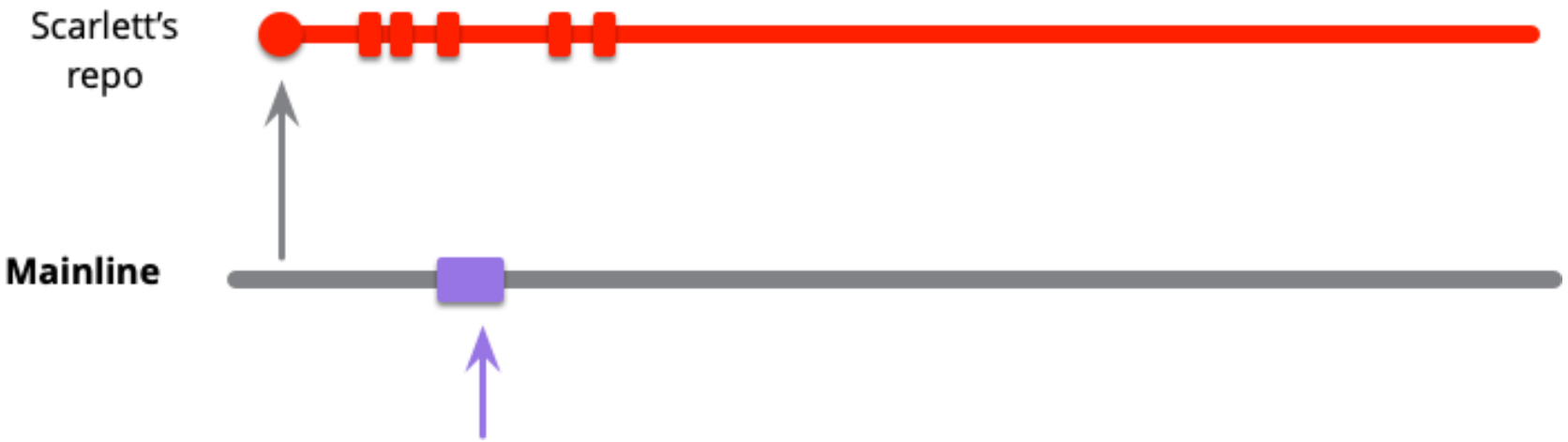
Mainline integration

Checkout:

- Pull + push
- Mainline: *healthy branch*
- Conflictos semánticos: *self-testing code*

Mainline integration

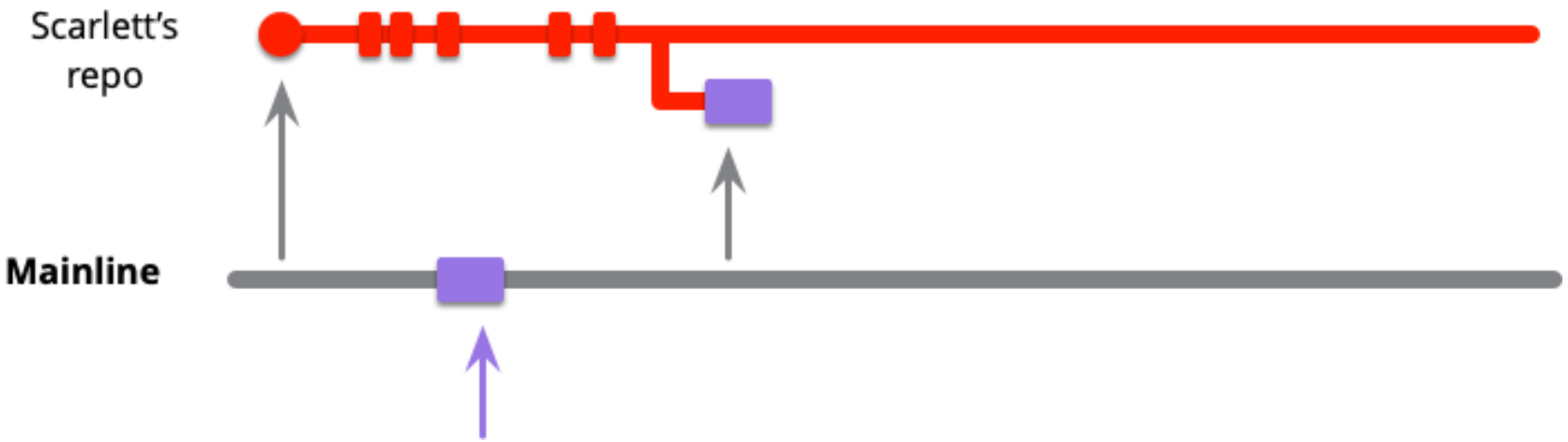
Update:



- Pull + push
- Mainline: *healthy branch*
- Conflictos semánticos: *self-testing code*

Mainline integration

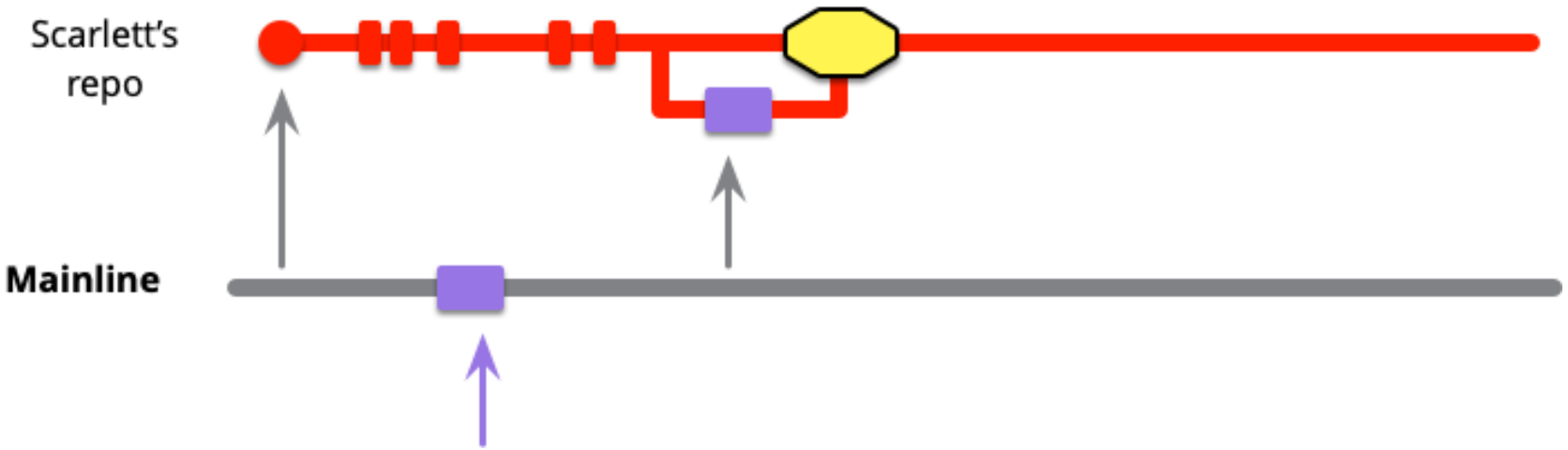
Pull:



- Pull + push
- Mainline: *healthy branch*
- Conflictos semánticos: *self-testing code*

Mainline integration

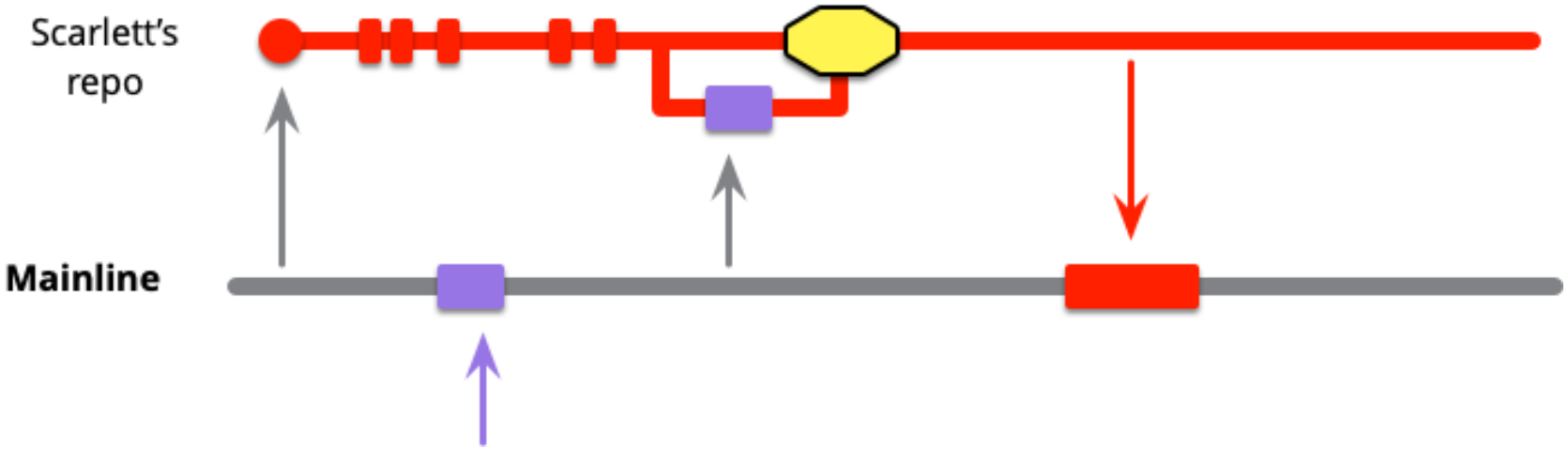
Merge:



- Pull + push
- Mainline: *healthy branch*
- Conflictos semánticos: *self-testing code*

Mainline integration

Integrate:

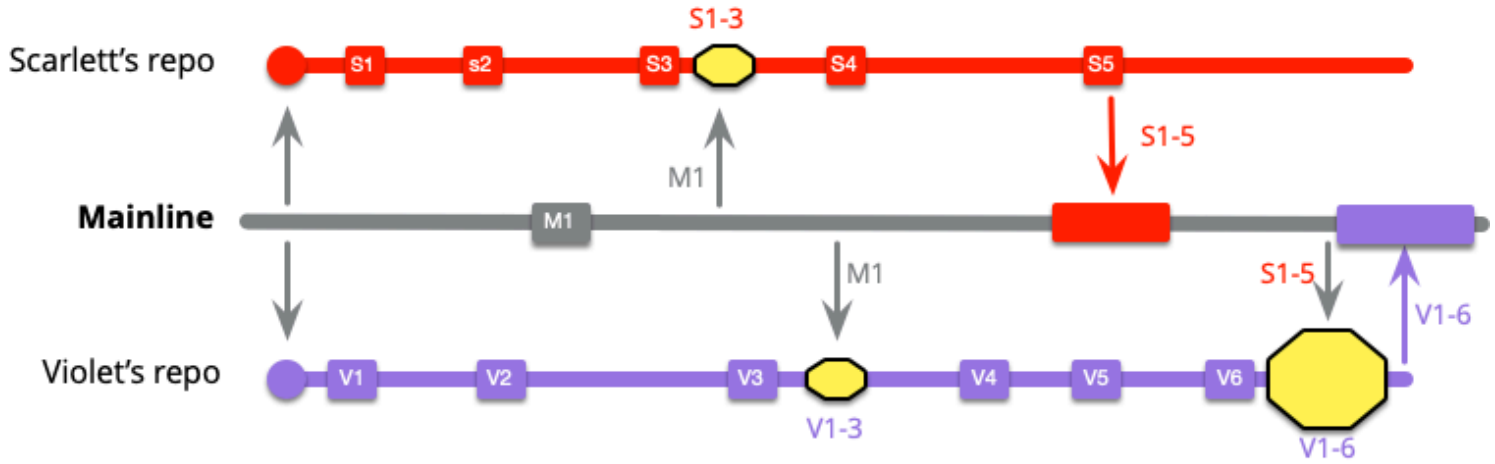


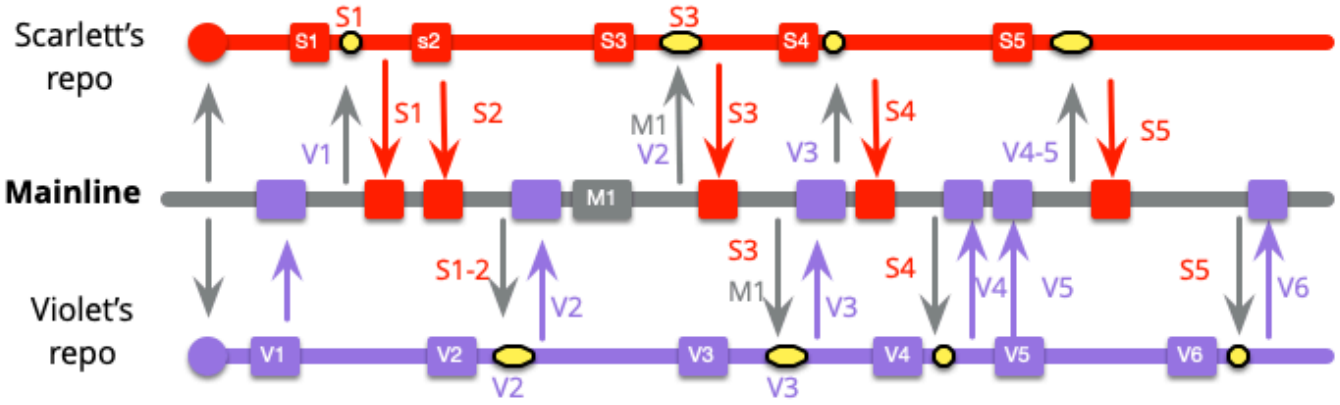
- Pull + push
- Mainline: *healthy branch*
- Conflictos semánticos: *self-testing code*

¿Cuál debe ser la frecuencia de integración?

Frecuencia de integración

Cantidad de trabajo



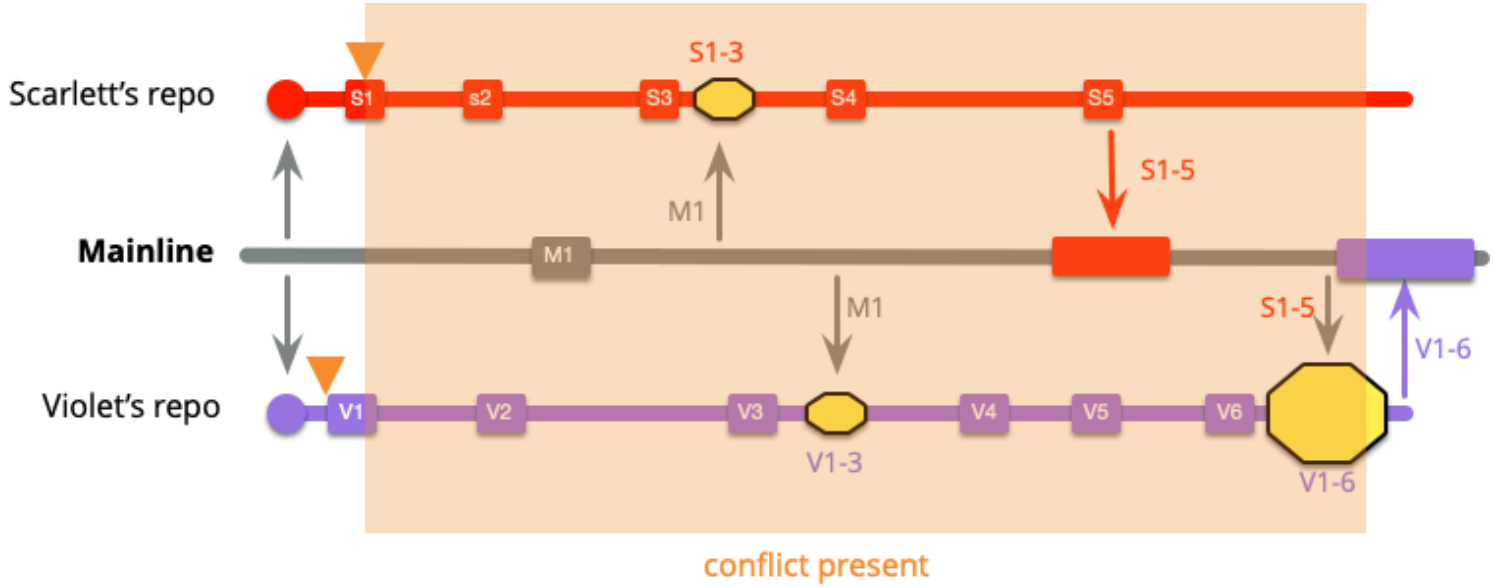


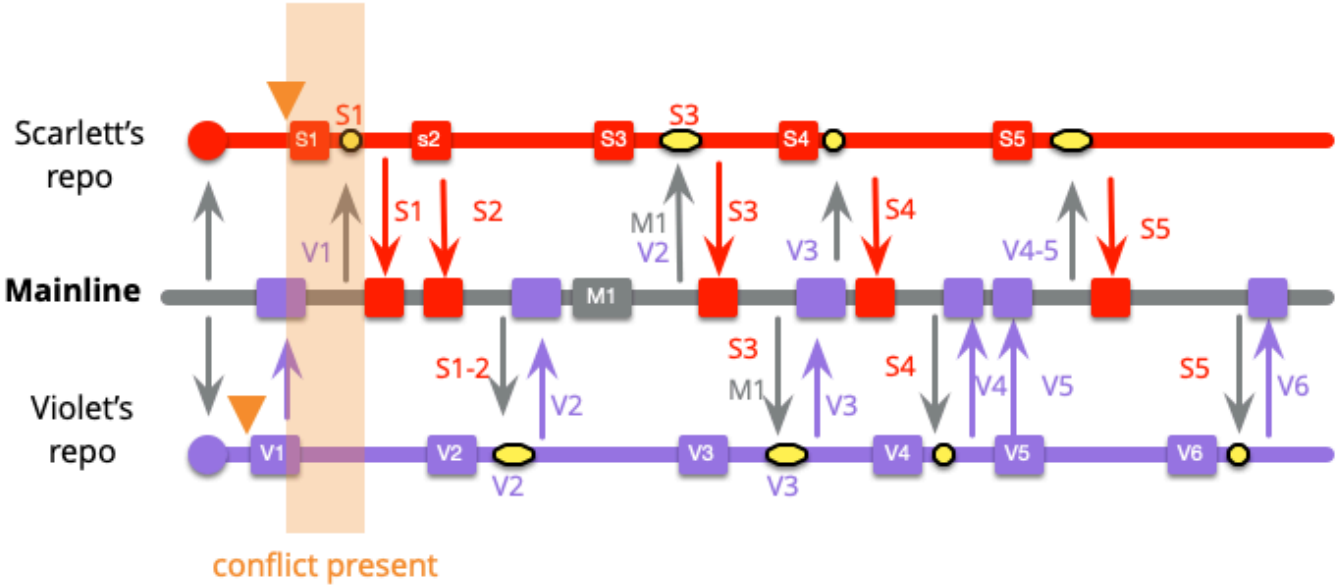
Frecuencia de integración

Riesgo de conflictos

¿Cuándo se detecta un posible conflicto en 1er. commit?

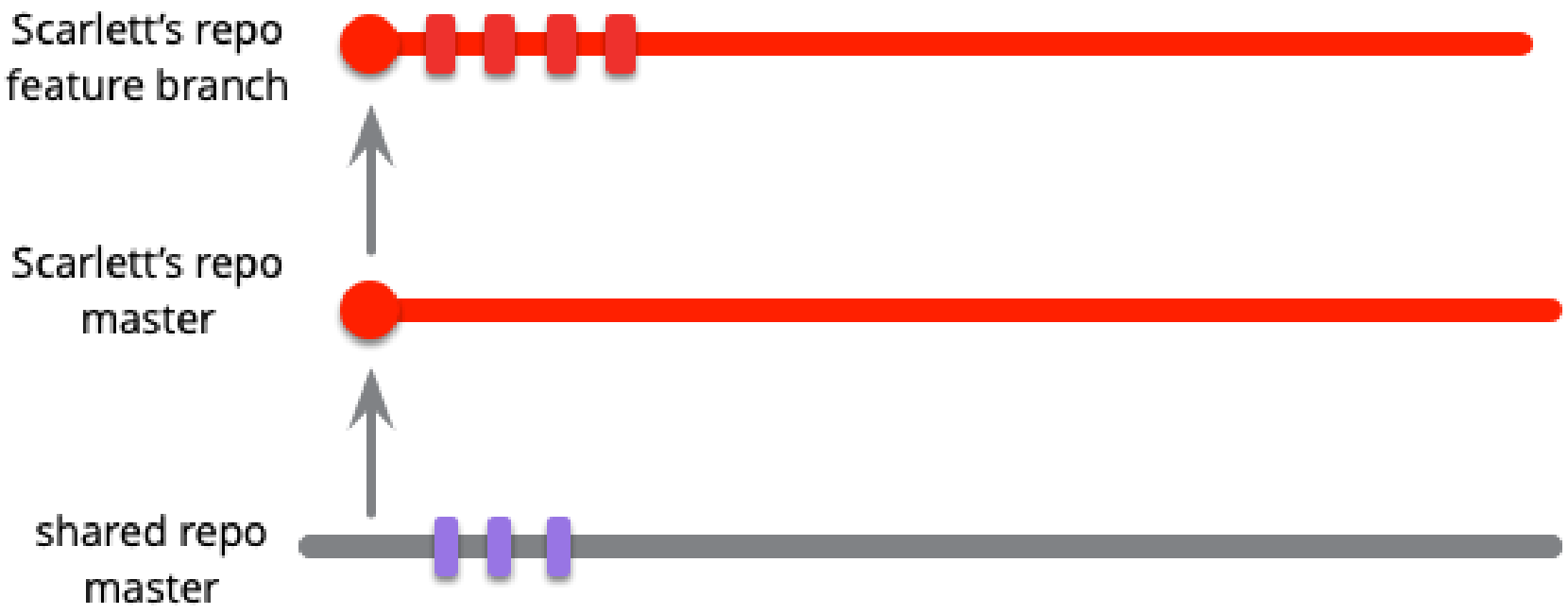
- baja frecuencia: merge final `S1` y `V1`
- alta frecuencia: primer merge





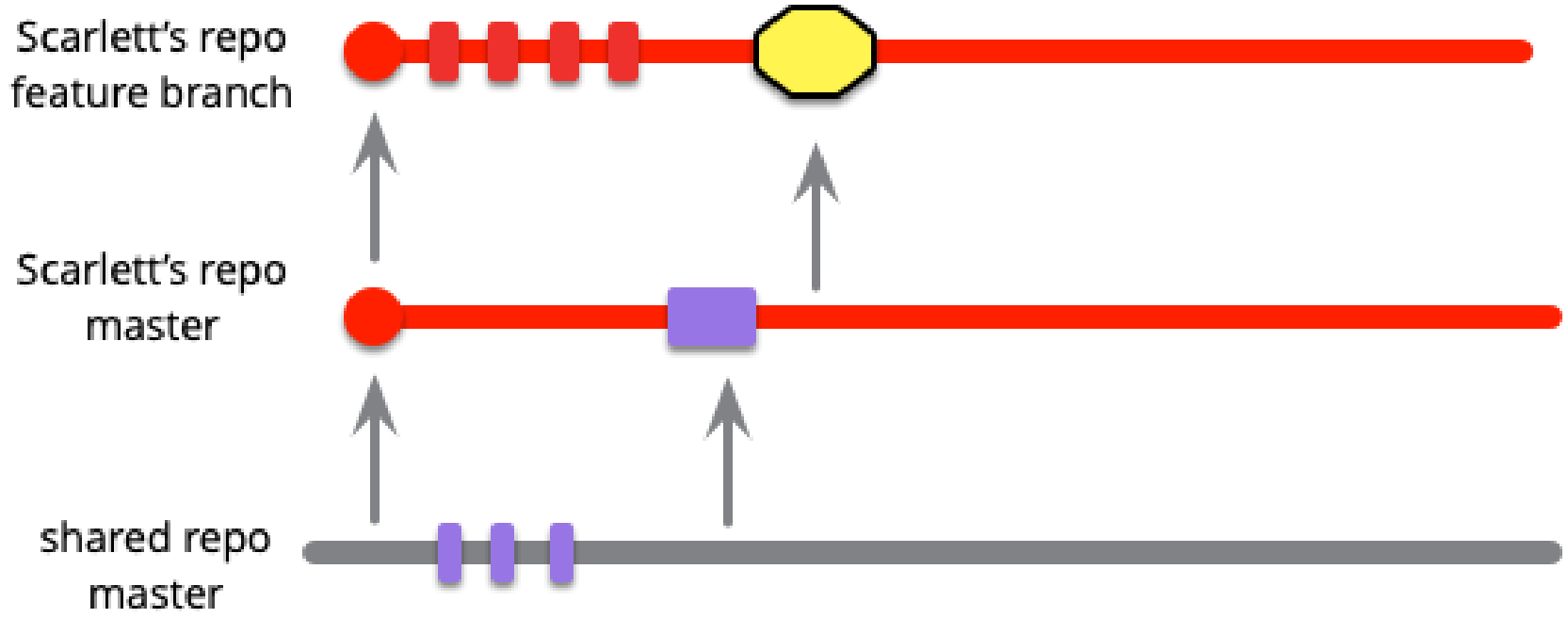
Feature branching

Branch:



Feature branching

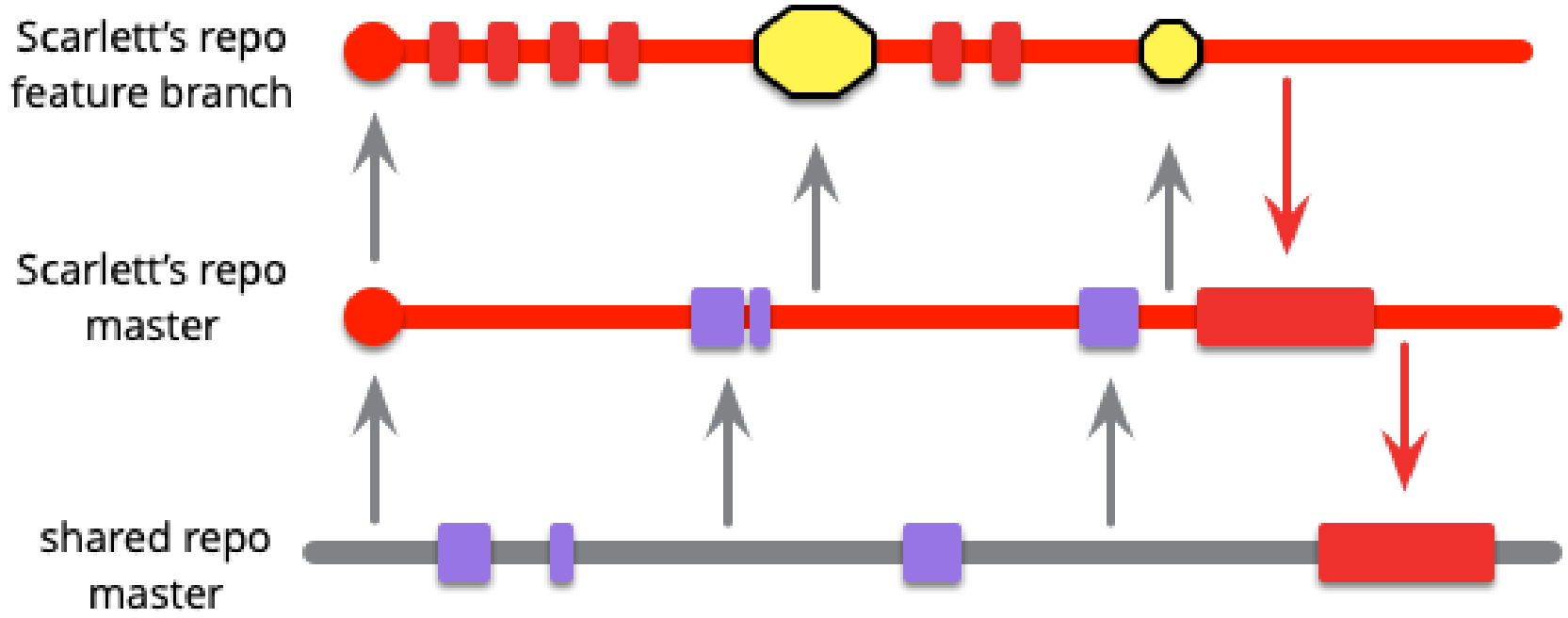
Pull:



- Llegan otros commits a la mainline

Feature branching

Integrate:



- Mainline integration

¿Es compatible el *feature branching* con la integración continua?

Feature branching y CI

- Depende de la frecuencia de integración
- Depende del tiempo que se tarda en completar una característica
- Continuous integration (CI)
  - Todos los commits de una característica van juntos
  - El código de cada característica siempre está en el producto
  - Feature flags para switch on/off
- Feature branching
  - No obliga a mantener ramas saludables
  - Puede disuadir de hacer refactoring (que introducen conflictos)

Feature Branching y Open Source

En proyectos open-source:

- Una o pocas personas como mantenedores y programadores
- Un grupo más grande de contribuidores (desconocidos para el mantenedor)
- Calidad de código discutible
- Incertidumbre total sobre el tiempo que dedicarán los contribuidores y su eficacia

En proyectos comerciales:

- Un equipo de personas conocidas y comprometidas a tiempo completo
- Expectativas fiables de la calidad del código y la capacidad de entrega
- Empleados remunerados y mayor control sobre el tiempo dedicado, estándares de codificación y hábitos del grupo

En proyectos open source, ¿qué se adapta mejor, *feature branching* o CI?

Patrones de entrega continua

Martin Fowler: *Patterns for managing source code branching*

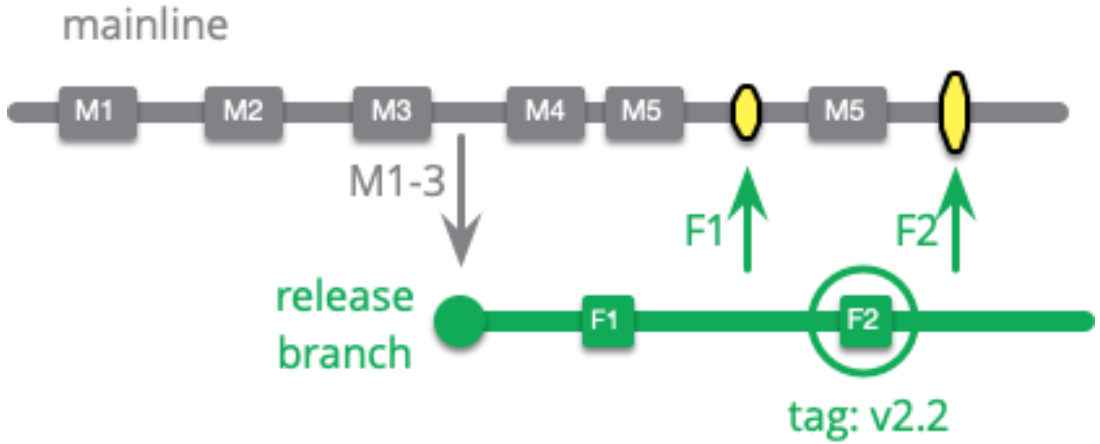
The path from mainline to production release

Entrega desde la *mainline*

- Release branch: Una rama que solo acepta los commits ya aceptados para una versión estable del producto, lista para su release
- Maturity branch: Una rama cuyo head marca la versión para producción de la codebase
- Environment branch: Contiene los commits necesarios para reconfigurar el producto para un entorno nuevo de ejecución
- Hotfix branch: Cada rama que captura el trabajo necesario para corregir un defecto urgente de producción
- Release train: Releases a intervalos regulares; desarrolladores eligen la suya
- Release-ready mainline: Mantener la mainline lo suficientemente saludable como para que el head pueda ir directamente a producción

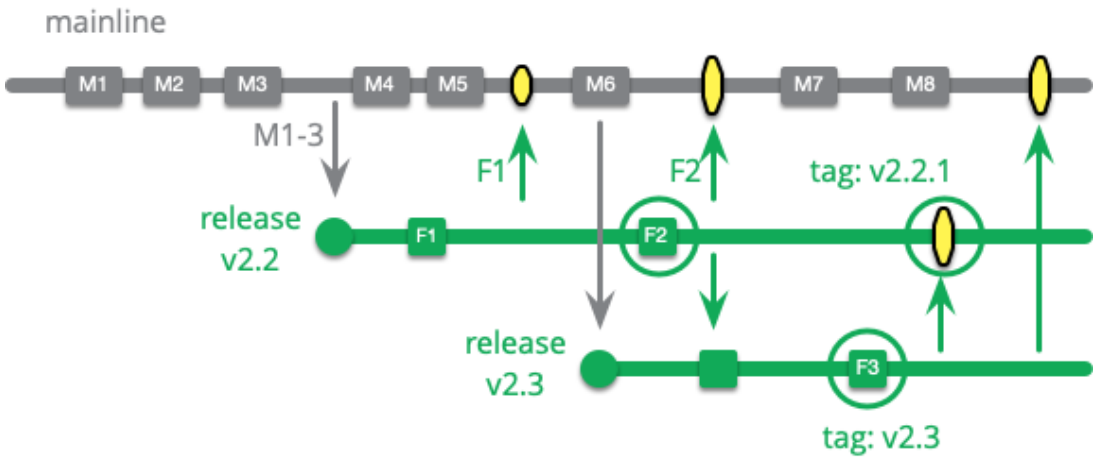
Release branch

- Release branch: única
- Maturity branch
- Environment branch
- Hotfix branch
- Release train
- Release-ready mainline



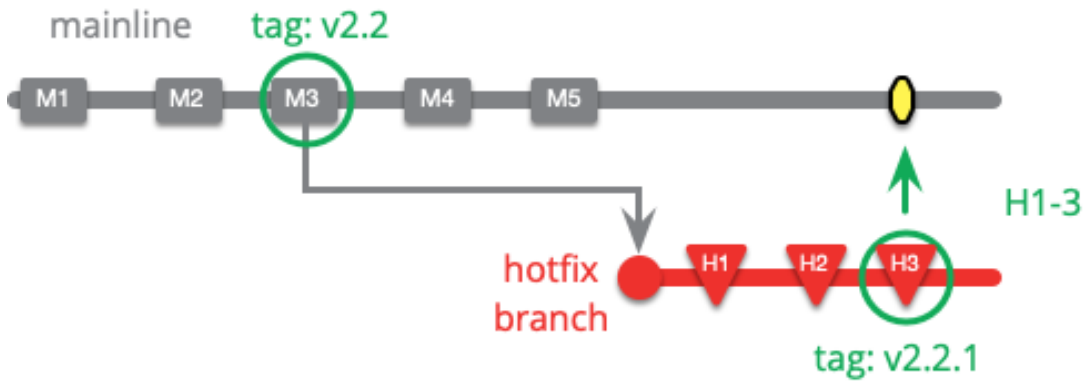
Release branch

- Release branch: múltiples
- Maturity branch
- Environment branch
- Hotfix branch
- Release train
- Release-ready mainline



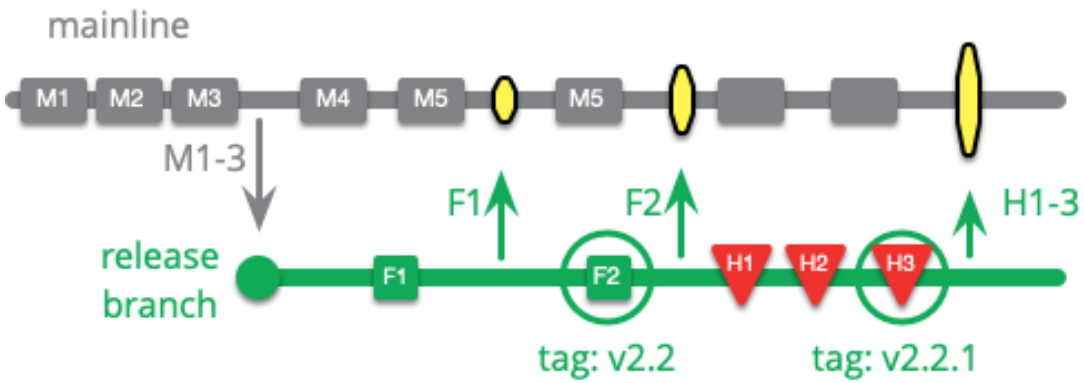
Hotfix branch

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train
- Release-ready mainline



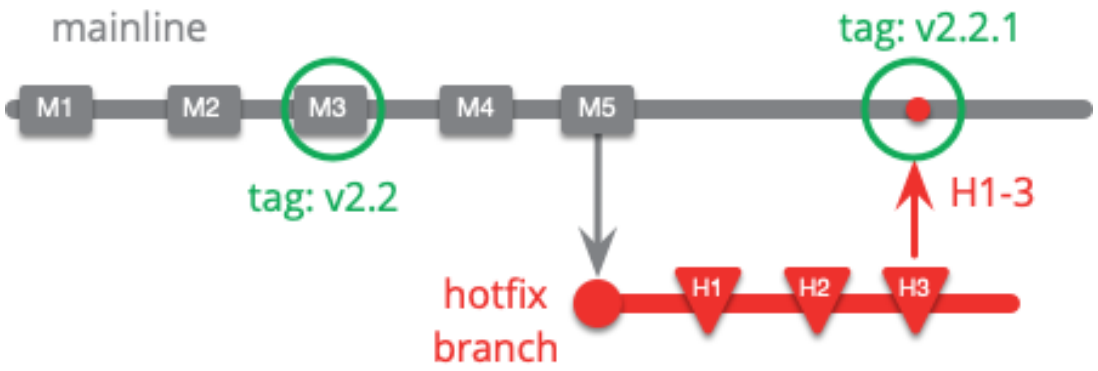
Hotfix branch

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch: con release branch
- Release train
- Release-ready mainline



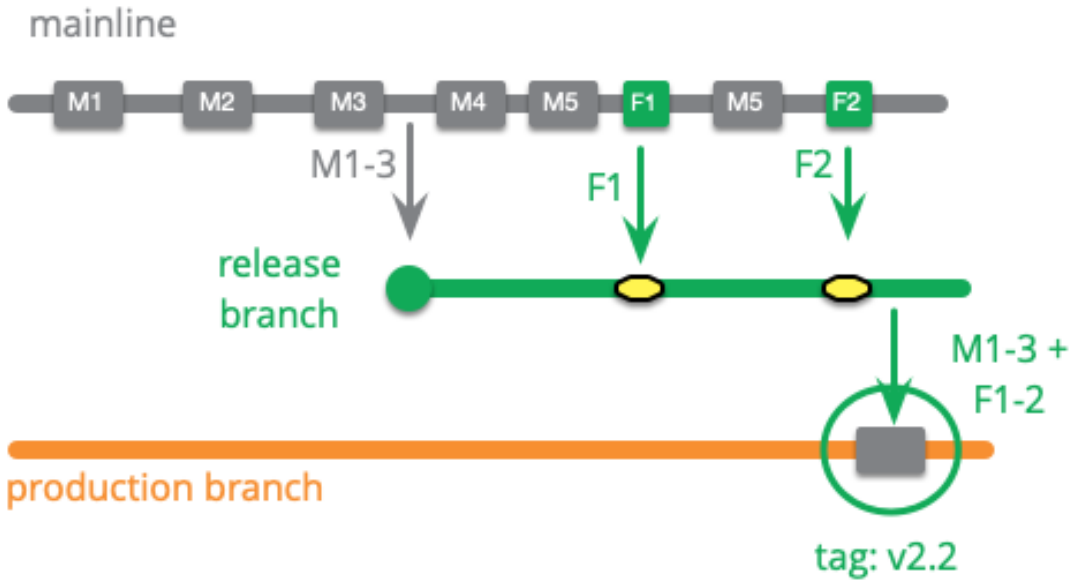
Hotfix branch

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch: desde mainline
- Release train
- Release-ready mainline



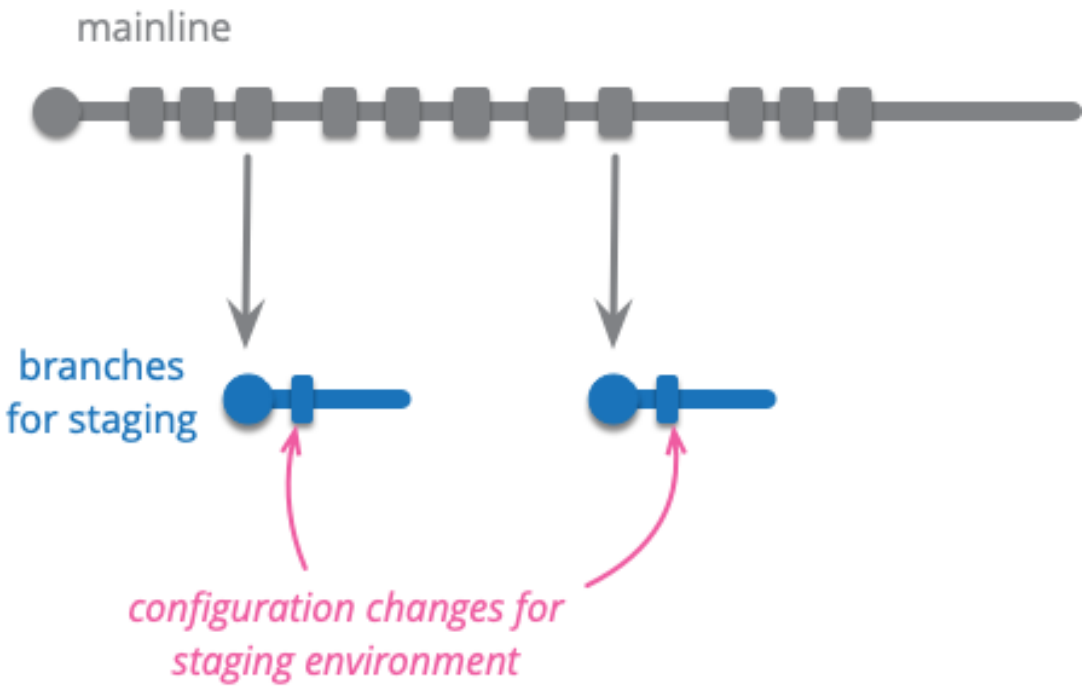
Maturity branch

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train
- Release-ready mainline



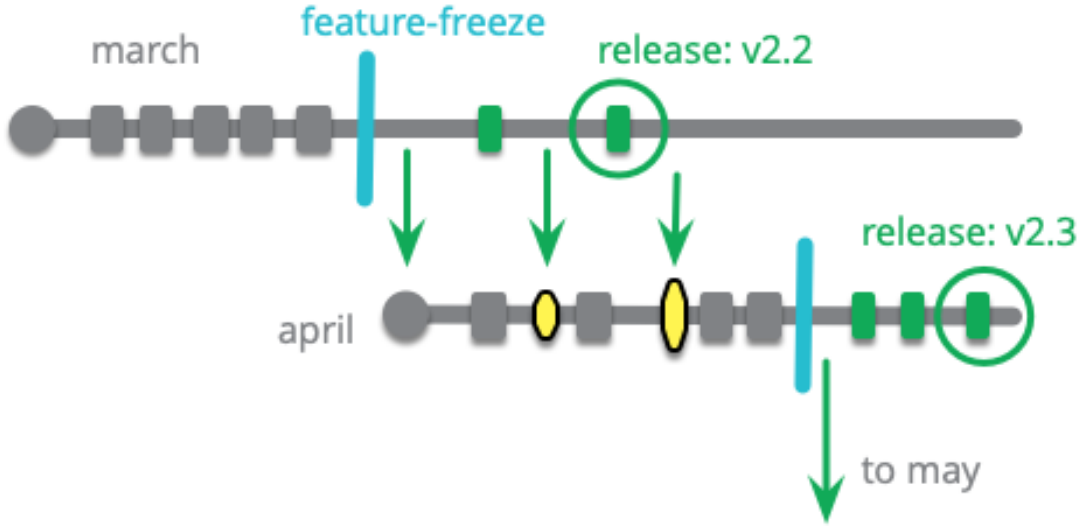
Environment branch

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train
- Release-ready mainline



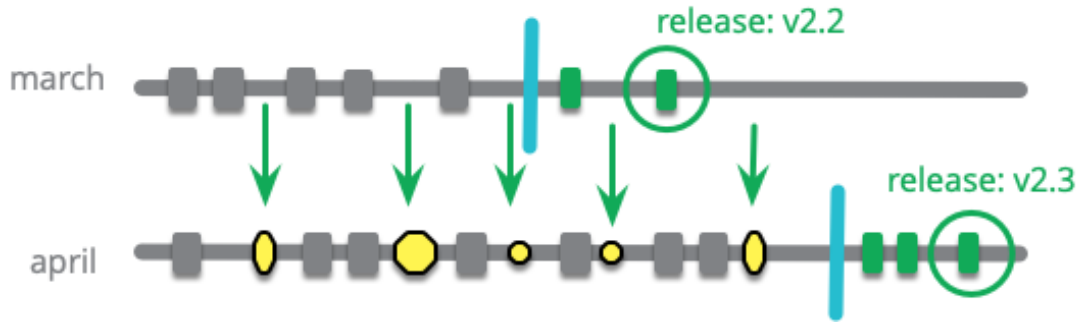
Release train

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train: múltiples
- Release-ready mainline



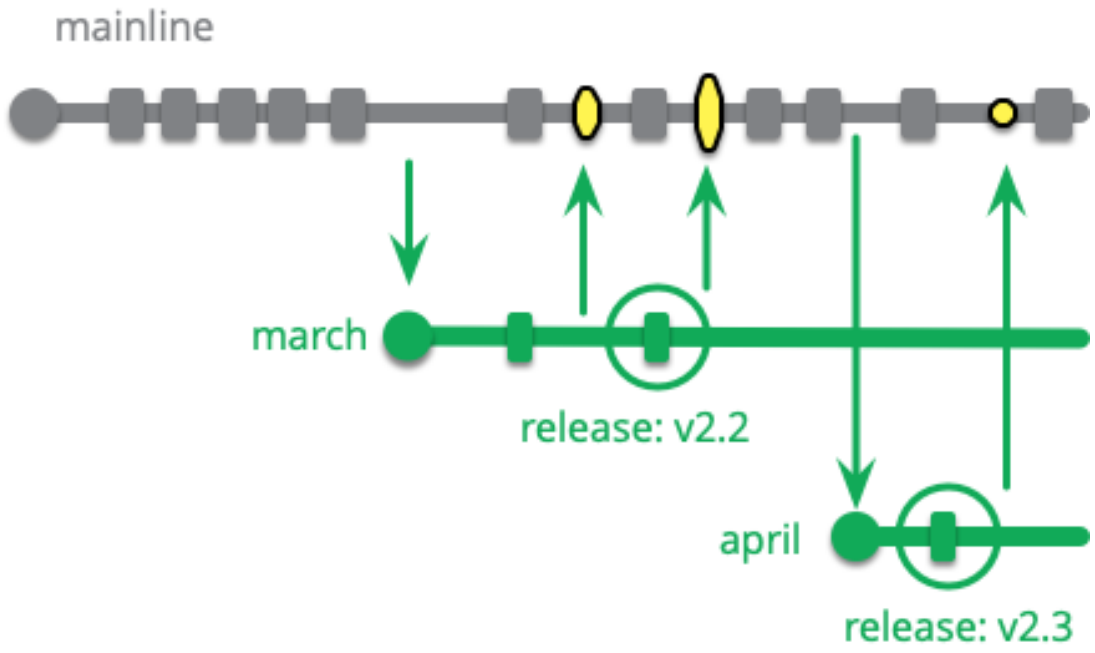
Release train

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train: trenes futuros
- Release-ready mainline



Release train

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train: desde mainline
- Release-ready mainline



Release-ready mainline

- Release branch
- Maturity branch
- Environment branch
- Hotfix branch
- Release train
- Release-ready mainline