Patrones de ramificación

Patrones Básicos

- Source Branching, cada desarrollador debe crear copias de la rama base y guardar todos los cambios en ellas.
- Mainline, una rama principal, accesible por todos los desarrolladores, que actúa como el estado actual del producto
- Healthy Branch, mantener una rama limpia, en la que en cada commit se ejecuten comprobaciones que aseguran que la rama no tiene defectos.

Patrones de Integración

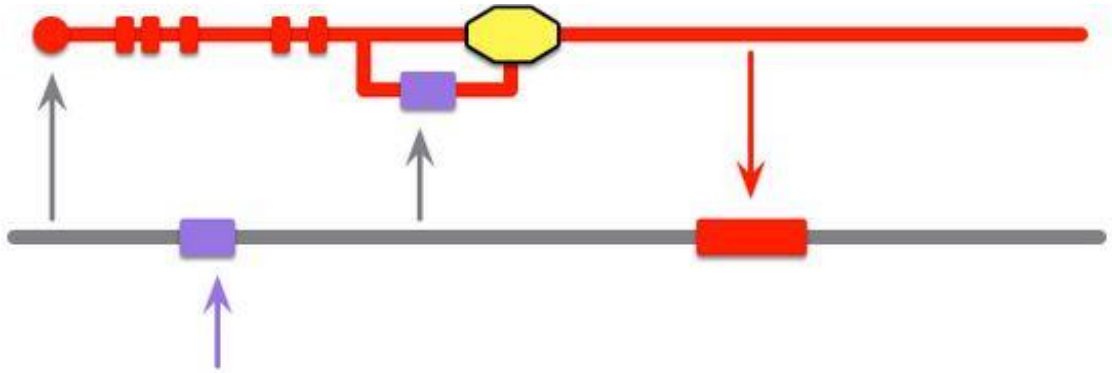
Estos patrones buscan las mejores estrategias para integrar las diferentes ramas de trabajo en un punto común:

- Mainline Integration
- Feature Branching
- Continuous Integration
- Pre-Integration review

Mainline Integration

Los desarrolladores integran su trabajo con el de la línea principal trayendo está a su entorno (pull), uniendo su trabajo (merge) y subiéndola de nuevo al espacio común (push).

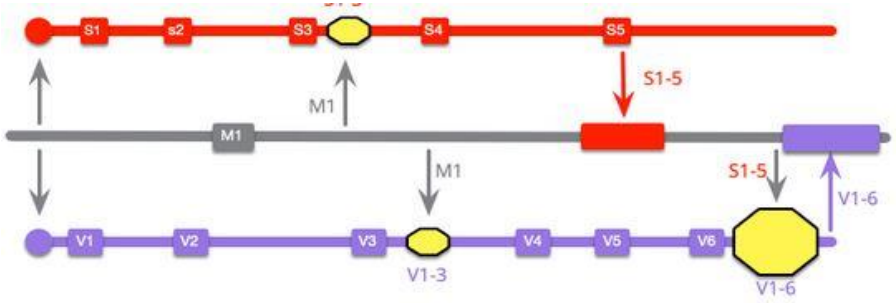
Esto simplifica mucho la integración y muestra más claramente el estado del proyecto.



Frecuencia de Integración

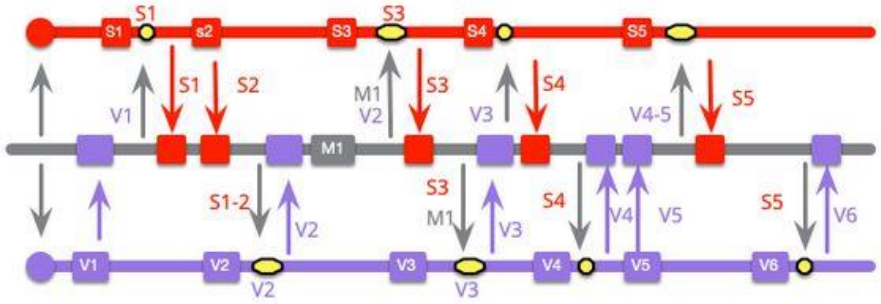
Este concepto hace referencia a la frecuencia con la que se integran los commits en la línea principal y el tamaño de estas integraciones.

Frecuencia baja:



Integraciones menos frecuentes, pero más difíciles y con más probabilidades de presentar errores graves

Frecuencia alta:



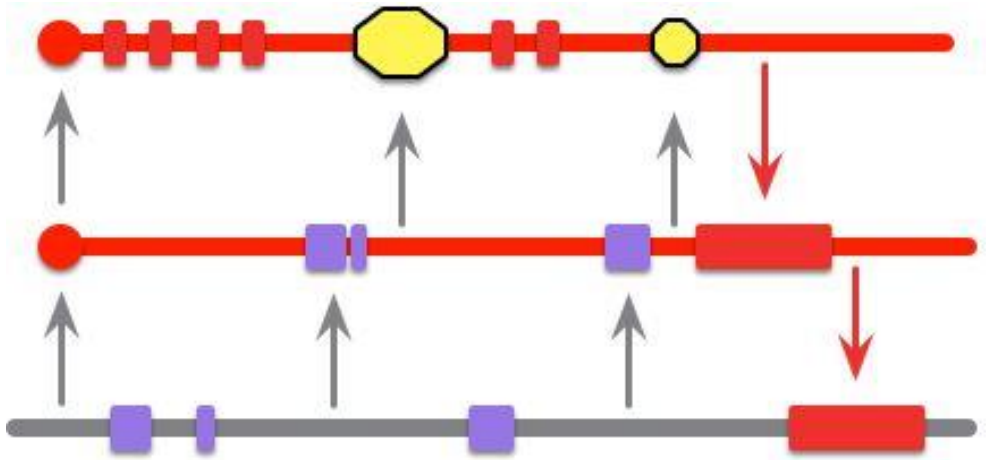
Integraciones más frecuentes y pequeñas, menos errores y se detectan antes.

Feature Branching

Cada tarea o característica del programa se desarrollará en su propia rama, y se integrará en la rama principal cuando se haya terminado.

Frecuencia de integración normalmente baja (con excepciones).

Muy utilizado en proyectos donde los colaboradores y la frecuencia con la que realizan tareas varía mucho.



Continuous Integration

Los desarrolladores deberán intentar integrar su trabajo en la rama principal tan pronto como tengan un commit en buen estado que puedan compartir. Como mucho un día por regla general.

Frecuencia de integración muy rápida.

Debido a esta integración continua habrá partes del proyecto a medio hacer por lo que habrá que tomar dos precauciones:

- Implementar tests para comprobar que dichas tareas a medio hacer no afecten al funcionamiento del proyecto
- Evitar el acceso a estas partes para que no se usen si el proyecto se va a mandar a producción.

Comparación

Los dos patrones anteriores son los más usados en la industria, en ocasiones es posible usarlos juntos si las tareas independientes son muy cortas, pero habitualmente hay que escoger uno de los dos.

Estas son las principales ventajas y desventajas de cada patrón.

Feature Branching

- ✓ All the code in a feature can be assessed for quality as a unit
- ✓ Feature code only added to product when feature is complete
- ✗ Less frequent merges

Continuous Integration

- ✓ Supports higher frequency integration than feature length
- ✓ Reduced time to find conflicts
- ✓ Smaller merges
- ✓ Encourages refactoring
- ✗ Requires commitment to healthy branches (and thus self-testing code)
- ✓ Scientific evidence that it contributes to higher software delivery performance

Pre-Integration Review

Todo commit que vaya a ser integrado en la rama principal debe ser revisado antes de poder aceptarse.

Si el miembro del equipo que revisa los commits detecta fallos en alguno de ellos deberá notificárselo al encargado de este, el cual hará las correcciones oportunas.

Funciona especialmente bien con el "Feature Branching" ya que este permite analizar cada característica del programa de forma completa y separada del resto.

También es posible con el patrón de "Continuous Integration" pero es más raro ya que resulta más largo y complicado.

Patrones de Producción

Estos patrones están pensados para mantener el producto en una versión always-releasable

- Release Branch
- Maturity Branch
- Environment Branch
- Hotfix Branch
- Release Train
- Release-Ready Mainline

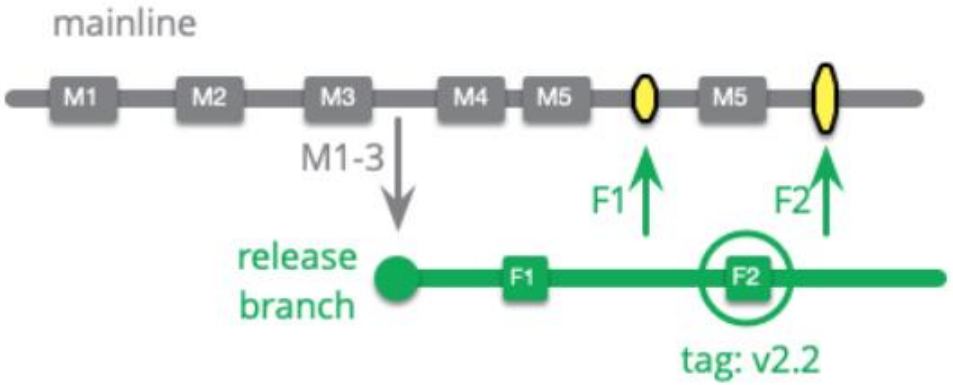
Release Branch

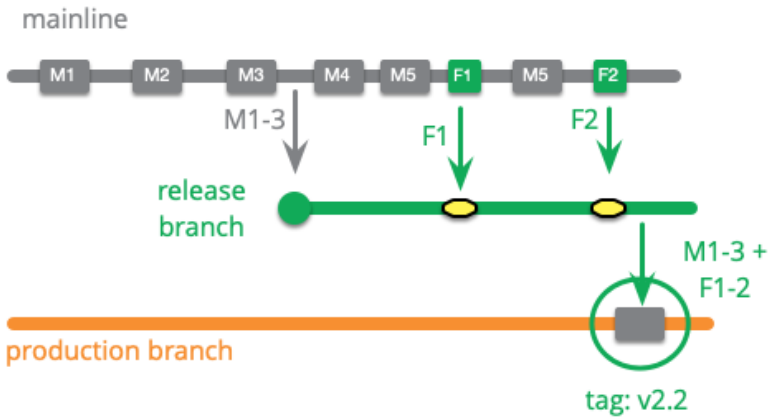
Descripción

Una rama que solo acepta commits para estabilizar una versión del producto listo para distribución.

Cuando usarlo

Recomendable cuando el equipo no es capaz de mantener su rama principal en un estado healthy





Maturity Branch

Descripción

Una rama que marca la última versión con un nivel de madurez del código base. Esta rama puede ser para producción, para lanzamiento, etc.

Cuando usarlo

Siempre que necesites mostrar rápidamente el historial de versiones.

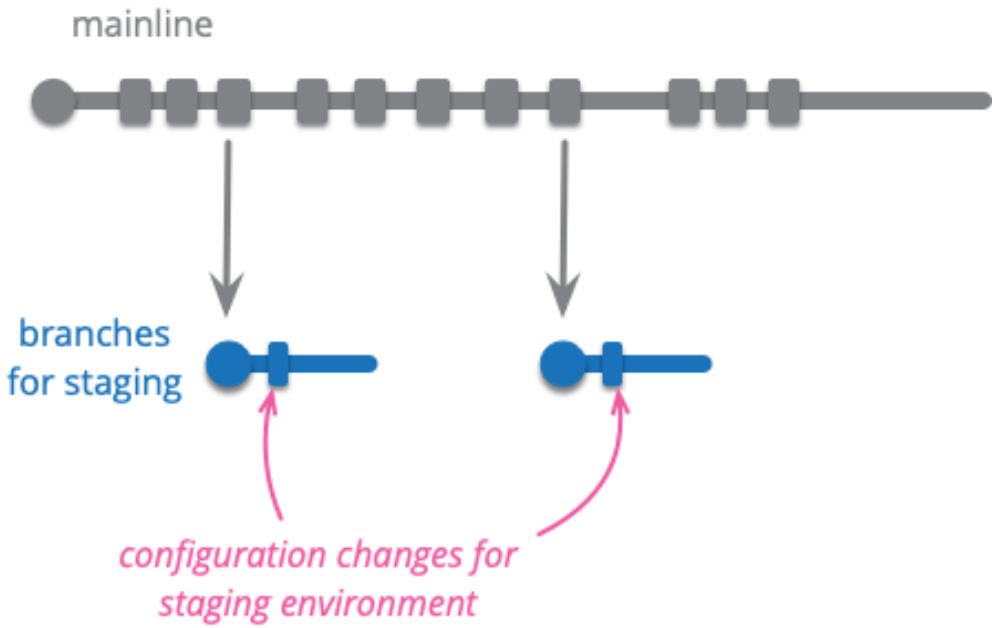
Environment Branch (often combined with Maturity Branch)

Descripción

Crear diferentes ramas para cada entorno de ejecución/desarrollo, y así almacenar la configuración necesaria para cada entorno.

Cuando usarlo

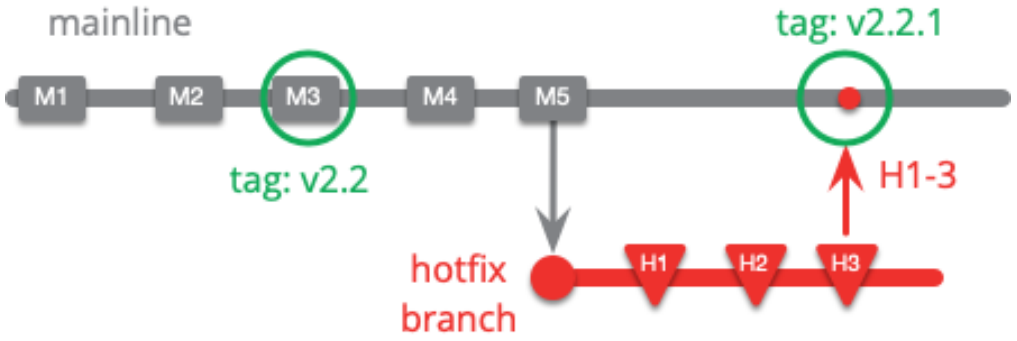
Si una aplicación necesita ejecutarse en diferentes entornos.



Hotfix Branch

Descripción

Crear una rama para un error que necesita ser rápidamente solucionado.



Cuando usarlo

Se usa sobre todo cuando el plazo de entrega está cerca y el equipo al estar bajo presión comete más errores que rompen la aplicación.

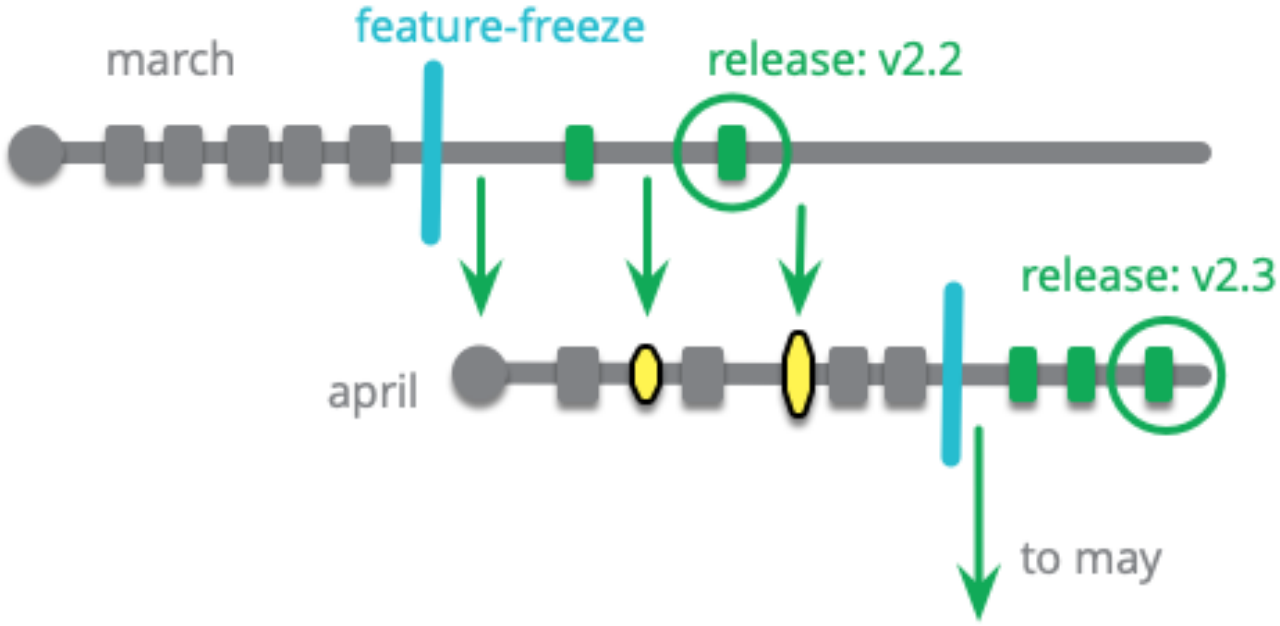
Release Train (usually used with Feature Branching)

Descripción

Crear ramas para diferentes intervalos de tiempos, como si fuera un horario regular de un tren.

Cuando usarlo

Cuando el proyecto tiene fechas de lanzamiento muy marcadas.



Release-Ready Mainline (make Mainline a Healthy Branch)

Descripción

Este patrón consiste en mantener la rama principal lo suficientemente estable y "saludable" para que siempre pueda ser puesta directamente en producción.

Cuando usarlo

Es recomendable usarlo siempre que puedas, pero, dependiendo del contexto, a veces no es recomendable (si tienes una frecuencia de integración baja)

Otros patrones

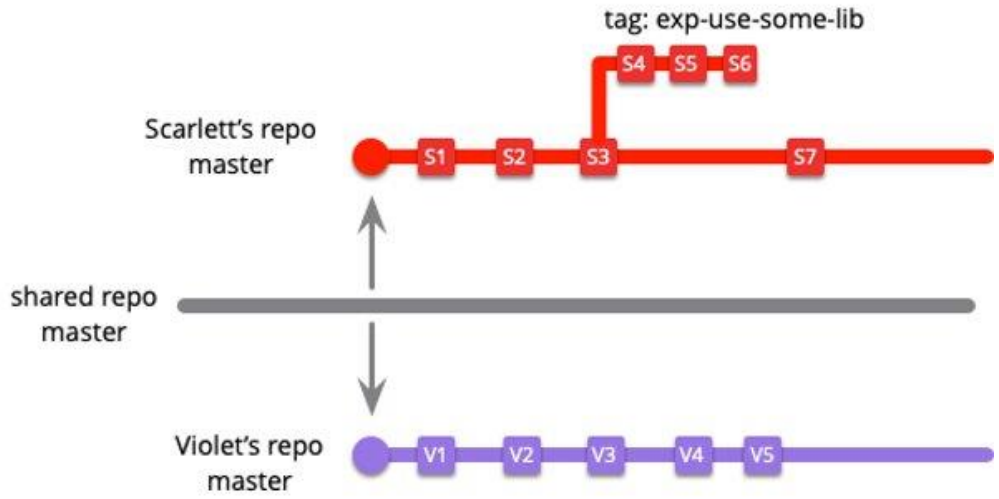
Algunos patrones que no se han mencionado anteriormente y que también es importante conocer serían los siguientes:

- Experimental Branch
- Future Branch
- Collaboration Branch
- Team Integration Branch

Experimental Branch

Descripción Su código posiblemente no llegue nunca a la mainline y será abandonado.

Cuando usarlo Siempre que se quiera probar algo y no estemos seguros de si acabará formando parte d ela mainline.



Future Branch

- Descripción Utilizada para cambios demasiado invasivos para los que las estrategias utilizadas normalmente no son útiles.

- Cuando usarlo Con cambios muy intrusivos en la arquitectura del sistema. No es muy común y no se usa si se aplica la Integración Continua.

Collaboration Branch

- Descripción Creada para que un desarrollador comparta el trabajo con otros miembros del equipo sin una integración.

- Cuando usarlo Cuanto más disminuye la frecuencia de integración, más útiles se vuelven. Si varias personas trabajan con un experimento, harán que esa rama se convierta en una Collaboration.

Team Integration Branch

- Descripción Permite que un sub equipo se integre entre si antes de integrarse con la línea principal.

- Cuando usarlo Cuando el trabajo está siendo desarrollado por un numero de desarrolladores que tiene sentido dividirlas en equipos separados.

Branching policies

- Git-Flow
- GitHub Flow
- Trunk-Based Development

Git-Flow

Mainline "develop"

Utiliza Feature Branching

Master como rama de Production Maturity branch

No se especifica que la mainline ha de ser Healthy branch

Hotfixes organizadas a traves de Hotfixes Branch

GitHub Flow

No se debe confundir con Git-Flow

No se utiliza Hotfix Branch

Pushear con regularidad

No hay integración con la línea principal hasta que se completa la función

Trunk-Based Development

Alternativa de política de ramificación a Git-Flow y GitHub Flow.

Todo el trabajo en mainline.

Equipos más grandes usan ramificacion de poca duración.

Release Branch o Release-Ready Mainline.