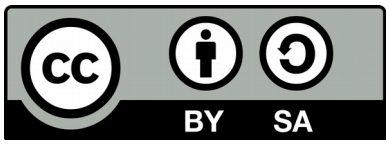
Fluho

Herramienta para la gestión de versiones y del flujo de trabajo durante el desarrollo de software

Documentación para versión 1.0.11-alpha

Autor: Juan José Castro Sotelo



Índice

1.1 Fluho....4
1.1.1 Características....4
1.1.2 Modelo de numeración de versiones....4
1.1.2.1 Esquema de numeración....4
1.1.2.1.1 Major, minor y rev....4
1.1.2.1.2 Status....4
1.1.2.1.3 Status_rev....5
1.1.2.1.4 Formatos reducidos....5
1.1.2.2 Nomenclatura de tipos de número de versión....5
1.1.2.2.1 Head....5
1.1.2.2.2 Cutting-edge....5
1.1.2.2.3 Maintenance-1....5
1.1.2.2.4 Maintenance-2....5
1.1.2.3 Política de cambio de numeración en componentes....5
1.1.2.3.1 Major....5
1.1.2.3.2 Minor....6
1.1.2.3.3 Rev....6
1.1.2.3.4 Status....6
1.1.2.3.5 Status_rev....7
1.1.2.3.6 Otras consideraciones....7
1.1.2.4 Ejemplo de progresión en el tiempo del número de versión....8
1.1.3 Modelo de organización del VCS....9
1.1.3.1 Esquema de etiquetas....9
1.1.3.2 Política de creación de etiquetas....9
1.1.3.2.1 Al comenzar el desarrollo....9
1.1.3.2.2 Al liberar una versión de prueba (alpha, beta o rc)....9
1.1.3.2.3 Al liberar una versión de producción (stable)....9
1.1.3.3 Esquema de ramas....9
1.1.3.4 Política de uso de ramas....10
1.1.3.4.1 Ramas de producción....10
1.1.3.4.1.1 cutting-edge (o master)....10
1.1.3.4.1.2 v{major}....10
1.1.3.4.1.3 v{major}.{minor}....10
1.1.3.4.2 Ramas de pruebas....10
1.1.3.4.3 Ramas de desarrollo....10
1.1.3.4.3.1 dev/v{major}.{minor}....11
1.1.3.4.3.2 dev/v{major}.{minor}/{feature}....11
1.1.3.4.4 ¿Por qué emplear la rama master como rama cutting-edge en Git?....11
1.1.3.5 Política de creación, fusionado y borrado de ramas....11
1.1.3.5.1 Al comenzar el desarrollo....11
1.1.3.5.2 Al comenzar el desarrollo de una nueva versión major....12
1.1.3.5.3 Al comenzar el desarrollo de una nueva versión minor de una major existente…12
1.1.3.5.4 Al comenzar el desarrollo de una nueva feature para una versión major.minor…12
1.1.3.5.5 Al finalizar el desarrollo de una nueva feature para una versión major.minor....12
1.1.3.5.6 Al liberar una versión de prueba (alpha, beta o rc)....12
1.1.3.5.6.1 Si número de versión tipo “cutting-edge” (head en major y en minor)....12

1.1.3.5.6.2 Si número de versión tipo “maintenance-1” (head solo en minor).....12
1.1.3.5.6.3 Si número de versión tipo “maintenance-2” (no head en minor).....12
1.1.3.5.7 Al liberar una versión de producción (stable) de una revisión más moderna.....12
1.1.3.5.7.1 Si número de versión tipo “cutting-edge” (head en major y en minor).....12
1.1.3.5.7.2 Si número de versión tipo “maintenance-1” (head solo en minor).....13
1.1.3.5.7.3 Si número de versión tipo “maintenance-2” (no head en minor).....13
1.1.4 Instalación de la herramienta.....14
1.1.4.1 Requisitos.....14
1.1.4.2 Proceso.....14
1.1.5 Configuración de la herramienta.....14
1.1.5.1 Rama cutting-edge.....14
1.1.5.2 Repositorio remoto.....15
1.1.5.2.1 Habilitar o deshabilitar el uso de un repositorio remoto.....15
1.1.5.2.2 Establecer qué repositorio remoto emplear.....15
1.1.5.2.3 Estrategia de sincronización con repositorio remoto.....15
1.1.5.3 Mensajes mostrados antes de liberar una versión.....16
1.1.5.3.1 Para liberación de versión para pruebas.....16
1.1.5.3.2 Para liberación de versión para producción.....16
1.1.5.4 Acciones extra al liberar una versión.....16
1.1.5.4.1 Para liberación de versión para pruebas.....16
1.1.5.4.2 Para liberación de versión para producción.....17
1.1.6 Ejecución de acciones comunes.....17
1.1.6.1 Iniciar el control de versiones con fluho en un repositorio (init).....17
1.1.6.2 Iniciar el desarrollo de una nueva versión (init).....17
1.1.6.3 Mostrar las versiones existentes (list).....17
1.1.6.4 Mostrar la última versión existente (list).....18
1.1.6.5 Mostrar información sobre la versión actual (status).....18
1.1.6.6 Cambiar a una versión (checkout).....18
1.1.6.6.1 Para desarrollo.....18
1.1.6.6.2 Para pruebas.....19
1.1.6.6.3 Para producción.....19
1.1.6.7 Desarrollar una funcionalidad de una versión.....20
1.1.6.8 Liberar una versión (release).....20
1.1.6.9 Cambiar el repositorio remoto (--remote).....20
1.1.6.10 Forzar trabajar solo en local (--noremote).....21
1.1.6.11 Replicar versiones locales en un repositorio remoto (sync).....21
1.1.7 Ejemplo de acciones ejecutadas sobre repositorio.....21
2. Heading 1.....27
2.1 Heading 2.....27
2.1.1 Heading 3.....27
2.1.1.1 Heading 4.....27
2.1.1.1.1 Heading 5.....27
2.1.1.1.1.1 Heading 6.....27

1.1 Fluho

1.1.1 Descripción

Fluho es una herramienta de línea de comandos para Linux y compatibles orientada a facilitar:

- la numeración de versiones: favorece el seguimiento de un esquema y política común
- el seguimiento de un flujo de desarrollo de software: guía el paso de versiones de desarrollo → versiones de prueba → versiones estables
- el mantenimiento de varias versiones en paralelo: crea ramas independientes y las funde y organiza de forma automática, inteligente y jerárquica al liberar versiones
- la gestión de la configuración: sirve como base para la integración en un sistema de *build* que permita seleccionar versiones concretas de cada desarrollo para formar la aplicación o sistema final

1.1.2 Características

- Permite mantener varias versiones en paralelo
- Rápida curva de aprendizaje, fácil y cómoda de usar en el día a día
- Permite mantener los cambios en un repositorio remoto compartido
- Permite acceder intuitiva y fácilmente al código deseado a desarrolladores, testers y usuarios finales, incluso sin tener fluho instalado
- Promueve el uso de buenas prácticas y el seguimiento de procedimientos, manteniendo toda la libertad posible
- Reduce la posibilidad de errores en la organización de las versiones y al liberar una versión
- Permite reconstruir todo el flujo seguido con observar el historial de comandos
- Mitiga la posibilidad de subida de cambios a las ramas de pruebas o de producción

1.1.3 Modelo de numeración de versiones

1.1.3.1 Esquema de numeración

El esquema de numeración de versiones es el siguiente:

Formato Base:	{*major*}.{*minor*}.{*rev*}-{*status*}
Formato Alternativo:	{*major*}.{*minor*}-{*status*}{*status_rev*}
Formato Reducido Tipo A:	{*major*}.{*minor*}.{*rev*}
Formato Reducido Tipo B:	{*major*}.{*minor*}
Formato Reducido Tipo C:	{*major*}

1.1.3.1.1 Major, minor y rev

Estos campos representan el número de versión principal, versión secundaria y revisión, respectivamente.

Contienen valores numéricos entre 0 y un límite indefinido.

Las cifras se muestran con el menor número de dígitos necesario. Por ejemplo, se emplea 0 en vez de 00.

Si no se desea utilizar alguno de estos campos, se establece a cero.

1.1.3.1.2 *Status*

Este campo representa el estado de la versión *major*.*minor*.

Contiene los valores *alpha*, *beta*, *rc* o *stable*.

En el caso de omitirse, se considera que es *stable*.

1.1.3.1.3 *Status_rev*

Este campo representa un valor numérico entre 1 y un límite indefinido.

La cifra se muestra con el menor número de dígitos necesario. Por ejemplo, se emplea 1 en vez de 01.

Únicamente es empleado en el formato alternativo de esquema de numeración, como contador del número de versiones *major*.*minor* con el mismo *status*.

1.1.3.1.4 *Formatos reducidos*

Los formatos reducidos sólo pueden emplearse si la versión referida existe en un *status stable*.

El emplear un formato reducido, siempre se estará haciendo referencia al último número de versión estable existente de los campos omitidos.

1.1.3.2 *Nomenclatura de tipos de número de versión*

1.1.3.2.1 *Head*

Un número de versión *major* es tipo "*head*" si es el mayor existente.

Un número de versión *minor* es tipo "*head*" si es el mayor existente de la *major* a la que pertenece.

1.1.3.2.2 *Cutting-edge*

Un número de version *major*.*minor* es tipo *"cutting-edge"* si es *head* en *major* y en *minor*.

1.1.3.2.3 *Maintenance-1*

Un número de version *major*.*minor* es tipo "*maintenance*-1" si es *head* solo en *minor*.

1.1.3.2.4 *Maintenance-2*

Un número de version *major*.*minor* es tipo "*maintenance*-2" si no es *head* en *minor*.

$$\underbrace{\underbrace{major\,.minor\,.\,\underbrace{revision}_{maintenance-2}}_{maintenance-1}}_{cutting-edge}$$

Illustration 1: Esquema de la nomenclatura de los tipos de número de versión según head

1.1.3.3 Política de cambio de numeración en componentes

1.1.3.3.1 Major

Se incrementa cuando se produce alguno de estos cambios respecto a la anterior versión *major stable*:

- Cambio en la interfaz de programación (API, CLI, ...) que rompe la compatibilidad.
- Rediseño del HMI que requiere del usuario un aprendizaje muy importante o que proporciona a este nuevas posibilidades muy relevantes.
- Modificación en la filosofía de funcionamiento que provoca cambios en la forma de uso o en los resultados obtenidos y rompe el soporte al paradigma empleado en la versión anterior.
- Cambios en la lista de funcionalidades o características que modifica el ámbito de aplicación del componente de forma representativa.

Un incremento en la versión *major* puede incluir cambios propios de incrementos en *minor* y *rev*.

1.1.3.3.2 Minor

Se incrementa cuando se produce alguno de estos cambios respecto a la anterior versión *minor* estable, pero no se produce ninguno que requiera el incremento de la versión *major*:

- Mejora de rendimiento o del consumo de recursos notable por el usuario o que permite la utilización del componente en modelos de hardware considerablemente menos potentes.
- Adición de nueva funcionalidad o característica que permite mejorar la aplicación del componente en un contexto que ya estaba soportado o en un nuevo contexto poco representativo.
- Reparo de defecto (error o vulnerabilidad) que impedía el funcionamiento aceptable de alguna característica, y/o comprometía la estabilidad o disponibilidad del sistema o de la aplicación, y/o permitía el acceso indebido al sistema o a información sensible y/o suponía un peligro personal grave.

Es posible saltar varias versiones *minor* de una vez si se producen varias de estar circunstancias o si los cambios son muy significativos pero no lo suficiente como para incrementar el número *major*.

Un incremento en la versión *minor* puede incluir cambios propios de incrementos en el número de *rev*.

1.1.3.3.3 Rev

Se incrementa cuando se produce alguno de estos cambios respecto a la anterior versión *rev*, pero no se produce ninguno que requiera el incremento de la versión *major* ni *minor*:

- Reescritura del código fuente.
- Modificación en la documentación.
- Mejora de rendimiento o del consumo de recursos no perceptible o apenas perceptible por el usuario y que no supone poder incluir el componente en modelos de hardware considerablemente menos potentes.
- Reparo de defecto que provocaba alguna discrepancia respecto a la especificación o estándar pero que no impedía el funcionamiento aceptable de ninguna característica, y/o comprometía únicamente la estabilidad o disponibilidad de funcionalidades no críticas, y/o permitía únicamente el acceso indebido a información o funcionalidades de naturaleza no sensible y/o suponía peligro personal leve.

Los incrementos del valor de este campo siempre deben ser consecutivos.

1.1.3.3.4 Status

Se establece a cada uno de los siguientes valores según sus respectivas condiciones. El avance de un estado a otro se realiza en este orden, nunca a la inversa, pudiendo saltarse estados:

- **alpha**: para referirse a una versión $X.Y$ cuyo desarrollo aún no contiene todas las funcionalidades y/o características planeadas para dicha versión.
- **beta**: para referirse a una versión $X.Y$ cuyo desarrollo ya está congelado en cuanto a funcionalidades y características pero que aún no está lista para su uso en producción porque no ha sido suficientemente probada y es probable que contenga defectos o incluso contiene defectos conocidos.

  También se conoce a esta versión como *preview*, *technical preview, technology preview* o prototipo.
- **rc**: para referirse a una versión $X.Y$ considerada como *beta* con alta probabilidad de resultar *stable*.
- **stable**: para referirse a una versión $X.Y$ que ha pasado las pruebas sin que aparezcan *bugs* significativos y está lista para su uso en producción.

  Cada estado *stable* de una versión $X.Y$ siempre corresponde a una revisión en estado rc. Esto implica que el estado *stable* debe ser idéntico al estado *rc* previo y que por cada versión *stable* siempre habrá una versión *rc* correspondiente.

  Ejemplo: Si la última versión *rc* es la *rc3*, hablar de la versión *stable1* debe ser lo mismo que hablar de la *rc3*, pues serán iguales bit a bit.

1.1.3.3.5 *Status_rev*

Únicamente es empleado en el formato alternativo de esquema de numeración, como contador del número de versiones *major*.*minor* con el mismo estado.

1.1.3.3.6 *Otras consideraciones*

- El número de versión `0.0.0-alpha` está reservado y deberá emplearse para indicar el comienzo del proyecto vacío dentro del sistema de control de versiones.
- A la hora de evaluar la gravedad de un defecto o de una vulnerabilidad de cara a decidir el número de versión a incrementar, únicamente deben tenerse en cuenta las consecuencias de la ocurrencia del error o de la explotación de la vulnerabilidad: pérdida de funcionalidad, pérdida de estabilidad, pérdida de disponibilidad, pérdida de confidencialidad o pérdida de control, es decir:
  - En tanto a un defecto, no se tiene en cuenta la probabilidad de ocurrencia de un error.
  - En tanto a una vulnerabilidad, no se tienen en cuenta los requisitos y características de un potencial agente amenaza (conocimientos, recursos, oportunidades, motivación y tamaño), la facilidad de descubrimiento y explotación de esta, el grado de conciencia de la existencia de esta ni la capacidad de rastreabilidad (detectar su explotación y por parte de quién).

  Esto es debido a que en un entorno de desarrollo, donde los componentes se emplean como parte de proyectos mayores, los factores que se tienen una naturaleza potencialmente dependiente del momento y de cómo se integre este dentro de un proyecto no se deben considerar. Un desarrollador debe poder conocer la importancia del cambio desde el punto de vista únicamente funcional a partir del número de versión y sin que esta se vea alterada por factores externos al componente.

1.1.3.4 *Ejemplo de progresión en el tiempo del número de versión*

Base	Alternativo	Tipo A	Tipo B	Tipo C	Observaciones
0.0.0-alpha	*0.0-alpha1*	N/D	N/D	N/D	El campo *status_rev* empieza a contar desde 1.
0.1.0-alpha	*0.1-alpha1*	N/D	N/D	N/D	El campo *status_rev* cuenta con respecto a la versión *major*.*minor* (y a cada *status*). *
1.0.0-alpha	*1.0-alpha1*	N/D	N/D	N/D	Si se sigue una metodología ágil, es posible que se avance de versión *minor* y/o *major* sin salir mientras del estado *alpha* o *beta*. *
1.0.1-alpha	*1.0-alpha2*	N/D	N/D	N/D	
...					El campo *rev* siempre es consecutivo.
1.0.23-alpha	*1.0-alpha24*	N/D	N/D	N/D	Recordar que *status_rev* cuenta desde 1 y no desde 0.
1.0.24-beta	*1.0-beta1*	N/D	N/D	N/D	El campo *status_rev* siempre cuenta con respecto a cada *status*. (y a cada versión *major*.*minor*)
1.0.25-beta	*1.0-beta2*	N/D	N/D	N/D	
1.0.26-rc	*1.0-rc1*	N/D	N/D	N/D	
1.0.26-stable	*1.0-stable1*	*1.0.26*	*1.0*	*1*	La versión *stable1* es siempre la última *rev* sin cambios.
1.0.27-rc	*1.0-rc2*	N/D	N/D	N/D	
1.0.27-stable	*1.0-stable2*	*1.0.27*	*1.0*	*1*	
~~*1.0.28-beta*~~	~~*1.0-beta3*~~	~~N/D~~	~~N/D~~	~~N/D~~	No retroceder a beta o alpha en estado de versión *major*.*minor*
1.1.0-rc	*1.1-rc1*	N/D	N/D	N/D	La 1ª versión 1.1 ha sido liberada directamente en estado *rc*. *
1.1.1-rc	*1.1-rc2*	N/D	N/D	N/D	
1.1.2-rc	*1.1-rc3*	N/D	N/D	N/D	
1.1.2-stable	*1.1-stable1*	*1.1.0*	*1.1*	*1*	
...					
1.1.123-rc	*1.1-rc124*	N/D	N/D	N/D	A cada versión *stable* le corresponde una versión *rc*.
1.1.123-stable	*1.1-stable122*	*1.1.123*	*1.1*	*1*	Recordar que *status_rev* cuenta desde 1 y no desde 0.
1.5.0-alpha	*1.5-alpha1*	N/D	N/D	N/D	El campo *minor* salta del 1 al 5, al incorporarse numerosas mejoras a la vez. *
1.5.1-beta	*1.5-beta1*	N/D	N/D	N/D	
1.5.2-rc	*1.5-rc1*	N/D	N/D	N/D	
1.5.2-stable	*1.5-stable1*	*1.5.2*	*1.5*	*1*	
2.0.0-alpha	*2.0-alpha1*	N/D	N/D	N/D	*
...					
2.0.2-stable	*2.0-stable1*	*2.0.2*	*2.0*	*2*	
2.1.0-beta	*2.1-beta1*	N/D	N/D	N/D	La primera versión 2.1 que se ha liberado en este caso se ha hecho directamente en el estado *beta*. *
2.1.1-rc	*2.1-rc1*	N/D	N/D	N/D	
2.1.1-stable	*2.1-stable1*	*2.1.1*	*2.1*	*2*	

** Nota: Las revisiones de versiones antiguas pueden continuar en paralelo a las de las más modernas.*

1.1.4 Modelo de organización del VCS

Se basa en el uso de etiquetas y ramas, principalmente de acuerdo a los números de las versiones.

1.1.4.1 *Esquema de etiquetas*

Las etiquetas se emplean para marcar la foto de estado de cada versión liberada, utilizando para ello cada una de las siguientes nomenclaturas, según aplique de acuerdo al tipo de la versión liberada:

- Formato Base: *v*{*major*}.{*minor*}.{*rev*}-{*status*}
- Formato Alternativo: *v*{*major*}.{*minor*}-{*status*}{*status_rev*}
- Formato Reducido Tipo A: *v*{*major*}.{*minor*}.{*rev*}

1.1.4.2 *Política de creación de etiquetas*

Nota: Esta política está pensada para su cumplimiento mediante el uso de herramientas automáticas. Si se realiza de forma manual, su cumplimiento completo puede resultar una tarea tediosa y propensa a errores. En tal caso, se recomienda simplificar la política ejecutando las acciones mínimas, marcadas con la señal "◄◄◄". Posteriormente, un *script* automático podrá deducir y ejecutar las acciones omitidas.

1.1.4.2.1 *Al comenzar el desarrollo*

- Creación de etiqueta *v0.0.0-alpha* ◄◄◄
- Creación de etiqueta *v0.0-alpha1*
- Creación de etiqueta *v0.0.0*

1.1.4.2.2 *Al liberar una versión de prueba (alpha, beta o rc)*

- Creación de la etiqueta *v*{*major*}.{*minor*}.{*rev*}-{*status*} ◄◄◄
- Creación de la etiqueta *v*{*major*}.{*minor*}-{*status*}{*status_rev*}

1.1.4.2.3 *Al liberar una versión de producción (stable)*

- Creación de la etiqueta *v*{*major*}.{*minor*}.{*rev*}-*stable* ◄◄◄
- Creación de la etiqueta *v*{*major*}.{*minor*}.{*rev*}

Solamente si la revisión que se está liberando es posterior a la última liberada anteriormente:

- Creación de la etiqueta *v*{*major*}.{*minor*}-*stable*{*status_rev*}

1.1.4.3 *Esquema de ramas*

El esquema de ramas a emplear es el siguiente:

- Ramas de producción (también conocidas como estables):
  - Para el proyecto en general: *cutting-edge* (o *master*)
  - Para cada versión *major*: *v*{*major*}
  - Para cada versión *major*.*minor*: *v*{*major*}.{*minor*}
- Ramas de pruebas (también conocidas como de validación o Q&A):
  - Para el proyecto en general: *test/cutting-edge*
  - Para cada versión *major*: *test/v*{*major*}
  - Para cada versión *major*.*minor*: *test/v*{*major*}.{*minor*}
- Ramas de desarrollo (también conocidas como de integración):

- Para cada versión *major*.*minor*: *dev/v*{*major*}.{*minor*}
- Para cada *feature* en una versión *major*.*minor*: *dev/v*{*major*}.{*minor*}/{*feature*} *(opcional)*

1.1.4.4 *Política de uso de ramas*

1.1.4.4.1 *Ramas de producción*

- Contienen las versiones estables liberadas.
- No admiten la subida de paquetes de cambios.
- Son empleadas por los responsables del sistema en producción y los usuarios, entre otros.

1.1.4.4.1.1 **cutting-edge (o master)**

Rama de versiones liberadas estables de tipo *cutting-edge*.

La cabecera de esta rama refleja la foto de estado de la versión estable liberada más avanzada del proyecto, es decir de la versión *stable* de tipo *cutting-edge* del proyecto.

En Git se empleará la rama *master* para realizar la función de la rama *cutting-edge* (por lo que no es necesario contar con una rama llamada *cutting-edge*).

1.1.4.4.1.2 **v{major}**

Rama de versiones liberadas estables de tipo *maintenance*-1 de esta *major*.

La cabecera de esta rama refleja la foto de estado de la versión *stable* más avanzada liberada de la versión *major*, es decir de la versión *stable* de tipo *maintenance*-1 de esta *major*.

Nótese que dicha versión podría coincidir con la versión *cutting-edge* del componente, en cuyo caso, para versiones estables, esta rama y la rama *cutting-edge* reflejarían la misma foto de estado.

1.1.4.4.1.3 **v{major}.{minor}**

Rama de versiones liberadas estables de tipo *maintenance*-2 de esta *major.minor*.

La cabecera de esta rama refleja la foto de estado de la revisión *stable* más avanzada liberada de la versión *major*.*minor*, es decir la revisión *stable* de tipo *maintenance*-2 de esta *major*.*minor*.

Nótese que dicha versión podría coincidir con la versión *maintenance*-1 de la versión *major*, en cuyo caso, para versiones estables, esta rama y la v{*major*} reflejarían la misma foto de estado, e incluso con la *cutting-edge* del proyecto, ocurriendo entonces lo mismo entre esta rama y la rama *cutting-edge*.

1.1.4.4.2 *Ramas de pruebas*

- Contienen las versiones no estables liberadas.
- No admiten la subida de paquetes de cambios.
- Son empleadas por los *testers* del equipo de validación y *Q&A*.
- Utilizan el prefijo "*test/*". (nota: la rama para versiones *cutting-edge*, debe llamarse *test/cutting-edge* y no *test/master*)
- El uso y características de cada una de ellas es igual que el de las ramas de producción, pero para versiones no estables (*alpha*, *beta* o *rc*).

1.1.4.4.3 *Ramas de desarrollo*

- Contienen los cambios para las revisiones todavía no liberadas.

- Admiten la subida de paquetes de cambios.
- Son empleadas por los desarrolladores.
- Utilizan el prefijo "*dev/*"

1.1.4.4.3.1 dev/v{major}.{minor}

Rama para ir introduciendo los cambios para la próxima revisión de la versión *major*.*minor* a liberar.

La cabecera de esta rama refleja la foto de estado con los últimos cambios desarrollados para tal revisión.

Empleada también por los sistemas de integración continúa para generaciones automáticas (*nightly*, …).

1.1.4.4.3.2 dev/v{major}.{minor}/{feature}

La creación de este tipo de rama es opcional.

Rama para desarrollar los cambios relacionados con una funcionalidad en concreto para una revisión futura de la versión *major*.*minor*.

La cabecera de esta rama refleja la foto de estado con los últimos cambios desarrollados para dicha funcionalidad.

1.1.4.4.4 *¿Por qué emplear la rama master como rama cutting-edge en Git?*

La rama *master* contiene el HEAD del repositorio y, por lo tanto, es la que se selecciona por defecto al clonar el repositorio.

Utilizando la rama *master* para contener las versiones *cutting-edge* estables, se consigue proporcionar de primeras la foto de estado de la versión estable más avanzada. Esto facilita que cualquier persona que desconozca la política de ramas pueda obtener y construir fácilmente dicha versión.

Si en la rama *master* se incluyesen las versiones liberadas no estables, o incluso los paquetes de cambios del desarrollo de la próxima versión a liberar, por un lado se estaría mitigando el impacto de un desarrollador mal informado que comenzase a trabajar en cambios a partir de una foto de estado de esta rama en lugar de una de desarrollo, por otro lado existiría el riesgo de construir por error una versión inestable y usarla en producción.

En general puede considerarse que la probabilidad de que se utilice por error la rama *master* para generar una versión para producción es menor que la de emplearla por descuido como base para nuevos desarrollos, dado que el primero es un procedimiento normalmente guiado o realizado por responsables instruidos, pero los efectos de este suceso serían potencialmente más graves que los de que alguien comience a desarrollar sin querer a partir de un estado antiguo.

Por ello se ha considerado como mejor opción, en general, la más conservadora, consistente en emplear la rama *master* como *cutting-edge*, en vez de como *test*/*cutting-edge*. En cualquier caso, migrar a utilizar la rama *master* como *test*/*cutting-edge*, si las circunstancias lo hiciesen preferible, sería posible y sencillo.

1.1.4.5 *Política de creación, fusionado y borrado de ramas*

Nota: Esta política está pensada para su cumplimiento mediante el uso de herramientas automáticas. Si se realiza de forma manual, su cumplimiento completo puede resultar una tarea tediosa y propensa a errores. En tal caso, se recomienda simplificar la política ejecutando las acciones mínimas, marcadas con la señal "◄◄◄". Posteriormente, un *script* automático podría deducir y ejecutar las acciones omitidas.

1.1.4.5.1 *Al comenzar el desarrollo*

- Se crea la rama *test/cutting-edge* a partir de la rama *cutting-edge* (*master*)
- Se crea la rama *v0* a partir de la rama *cutting-edge*
- Se crea la rama *test/v0* a partir de la rama *v0*

- Se crea la rama *v0.0* a partir de la rama *v0*
- Se crea la rama *test/v0.0* a partir de la rama *v0.0*
- Se crea la rama *dev/v0.0* a partir de la rama *test/v0.0* ◄◄◄ (a partir de la rama *master*)

1.1.4.5.2 *Al comenzar el desarrollo de una nueva versión major*

- Se crea la rama *v*{*major*} a partir de la rama *cutting-edge (master)*
- Se crea la rama *test/v*{*major*} a partir de la rama *v{major}*
- Se crea la rama *v*{*major*}.*{minor}* a partir de la rama *v*{*major*}
- Se crea la rama *test/v*{*major*}.*{minor}* a partir de la rama *v*{*major*}.*{minor}*
- Se crea la rama *dev/v*{*major*}.*{minor}* a partir de la rama *test/v*{*major*}.*{minor}* ◄◄◄ (desde *master*)

1.1.4.5.3 *Al comenzar el desarrollo de una nueva versión minor de una major existente*

- Se crea la rama *v*{*major*}.{*minor*} a partir de la rama *v*{*major*}
- Se crea la rama *test/v*{*major*}.{*minor*} a partir de la rama *v*{*major*}.{*minor*}
- Se crea la rama *dev/v*{*major*}.{*minor*} a partir de la rama *test/v*{*major*}.{*minor*} ◄◄◄ (desde *master*)

1.1.4.5.4 *Al comenzar el desarrollo de una nueva feature para una versión major.minor*

- Se crea la rama *dev/v*{*major*}.{*minor*}/{*feature*} a partir de la rama *dev/v*{*major*}.{*minor*} ◄◄◄

1.1.4.5.5 *Al finalizar el desarrollo de una nueva feature para una versión major.minor*

- Se fusiona la rama *dev/v*{*major*}.{*minor*}/{*feature*} con rama *dev/v*{*major*}.{*minor*} sin *fast-forward* ◄◄◄
- Se elimina la rama *dev/v*{*major*}.{*minor*}/{*feature*} ◄◄◄

1.1.4.5.6 *Al liberar una versión de prueba (alpha, beta o rc)*

1.1.4.5.6.1 Si número de versión tipo “cutting-edge” (head en major y en minor)

- Se fusiona la rama *dev/v*{*major*}.{*minor*} con la rama *test/v*{*major*}.{*minor*} con *fast-forward*
- Se fusiona la rama *test/v*{*major*}.{*minor*} con la rama *test/v*{*major*} con *fast-forward*
- Se fusiona la rama *test/v{major}* con la rama *test/cutting-edge* con *fast-forward*

1.1.4.5.6.2 Si número de versión tipo “maintenance-1” (head solo en minor)

- Se fusiona la rama *dev/v*{*major*}.{*minor*} con la rama *test/v*{*major*}.{*minor*} con *fast-forward*
- Se fusiona la rama *test/v*{*major*}.{*minor*} con la rama *test/v*{*major*} con *fast-forward*

1.1.4.5.6.3 Si número de versión tipo “maintenance-2” (no head en minor)

- Se fusiona la rama *dev/v*{*major*}.{*minor*} con la rama *test/v*{*major*}.{*minor*} con *fast-forward*

1.1.4.5.7 *Al liberar una versión de producción (stable) de una revisión más moderna*

IMPORTANTE: No realizar ninguna fusión de ramas si se está liberando a producción una revisión anterior de la *v*{*major*}.{*minor*} a otra liberada previamente (por ejemplo, si se está liberando la *v1.2.34-stable* cuando ya se liberó anteriormente la *v1.2.99*). Tan sólo crear las etiquetas correspondientes (*v*{*major*}.{*minor*}.{*rev*}-*stable*, ...).

Recuérdese que la revisión liberada para producción debe coincidir con la revisión de tipo *rc* liberada para pruebas.

1.1.4.5.7.1 Si número de versión tipo “cutting-edge” (head en major y en minor)

- Se fusiona hasta la etiqueta *v*{*major*}.{*minor*}.{*rev*}-*rc* con la rama *v*{*major*}.{*minor*} con *fast-forward*
- Se fusiona la rama *v*{*major*}.{*minor*} con la rama *v*{*major*} con *fast-forward*

- Se fusiona la rama *v{major}* con la rama *cutting-edge* con *fast-forward*

1.1.4.5.7.2 Si número de versión tipo “maintenance-1” (head solo en minor)

- Se fusiona hasta la etiqueta *v*{*major*}.{*minor*}.{*rev*}-*rc* con la rama *v*{*major*}.{*minor*} con *fast-forward*
- Se fusiona la rama *v*{*major*}.{*minor*} con la rama *v*{*major*} con *fast-forward*

1.1.4.5.7.3 Si número de versión tipo “maintenance-2” (no head en minor)

- Se fusiona hasta la etiqueta *v*{*major*}.{*minor*}.{*rev*}-*rc* con la rama *v*{*major*}.{*minor*} con *fast-forward*

1.1.5 Instalación de la herramienta

1.1.5.1 *Requisitos*

- Git v2.7.4 (versión incluida en Ubuntu 16.04) *//TODO Probar en versiones anteriores*
- Bash 4.3.46 (versión incluida en Ubuntu 16.04) *//TODO Probar en versiones anteriores*
- Grep
- Sed
- Awk
- Wc
- Sort
- Tail

1.1.5.2 *Proceso*

Primeramente debe obtenerse una copia del código de Fluho, si no se tiene todavía. Para ello, realizar un clonado del repositorio de Git en un directorio temporal, por ejemplo `/tmp/fluho`, ejecutando un comando como el siguiente:

```
$ git clone https://github.com/swans-oss/fluho.git /tmp/fluho
```

En el momento de la escritura de esta versión del manual, el código fuente utilizable más moderno se encuentra en la rama de pruebas de la versión 1.0, ya que aún no hay liberada ninguna versión estable, por lo que habrá que situarse en la rama '`test/v1`':

```
$ cd /tmp/fluho
```

```
$ git checkout test/v1
```

Desde el directorio `src` del paquete de distribución o repositorio de Fluho, ejecutar, con permisos de administrador, el script `install`:

```
$ cd /tmp/fluho/src
```

```
# ./install
```

Esto instalará por defecto en la ruta `/opt/fluho`.

Si se desea instalar en otro directorio distinto del por defecto, utilizar el argumento `--destdir` para ello:

```
# ./install --destdir=/directory/to/install/fluho
```

1.1.6 Configuración de la herramienta

1.1.6.1 *Rama cutting-edge*

Es posible especificar el nombre de la rama que actuará como cutting-edge (la que contiene la versión estable liberada más avanzada del proyecto).

Para ello editar el código fuente del fichero `fluho.user` y modificar el nombre contenido en la variable `CUTTINGEDGE_BRANCH`.

Por defecto, se encontrará la siguiente configuración:

```
CUTTINGEDGE_BRANCH="master"
```

1.1.6.2 *Repositorio remoto*

Cuando se trabaja con un VCS distribuido, como Git, es posible trabajar únicamente en el repositorio local o compartir los cambios en un repositorio remoto que actúe de central.

Fluho permite utilizar un repositorio remoto para compartir y coordinar centralizadamente los cambios sobre las versiones entre todos los integrantes del equipo de desarrollo. Esto significa que si un integrante del equipo, por ejemplo, crea una nueva versión o libera una revisión, todos los demás integrantes también tendrán constancia de ella.

1.1.6.2.1 *Habilitar o deshabilitar el uso de un repositorio remoto*

Por defecto, Fluho está configurado para trabajar en un entorno colaborativo donde se repliquen todos los cambios sobre versiones en un repositorio remoto compartido.

Esta funcionalidad está habilitada en el fichero `fluho.user` en la variable '`USE_REMOTE`':

```
USE_REMOTE=true
```

A pesar de ello, si no hay un repositorio remoto configurado en el VCS, Fluho trabajará automáticamente en local, aunque se mostrará un mensaje de advertencia.

Esta replicación de las acciones sobre versiones en el repositorio remoto se puede desactivar de forma global, editando la configuración del programa, o de forma limitada a un comando, mediante un argumento.

Para hacerlo de forma global, editar el código fuente del archivo `fluho.user` y establecer la variable '`USE_REMOTE`' a '*false*':

```
USE_REMOTE=false
```

Aunque la variable '`USE_REMOTE`' esté establecida a '*false*', se puede forzar el uso del repositorio remoto de forma puntual en un comando utilizando el argumento `--remote {remote_alias}`.

1.1.6.2.2 *Establecer qué repositorio remoto emplear*

En el caso de que en el VCS existan varios repositorios remotos configurados, se puede especificar cuál usar. Para ello editar el código fuente del archivo `fluho.user` y establecer el contenido de variable '`REMOTE_ALIAS`' al alias de repositorio remoto deseado.

Por defecto este alias es 'origin', que coincide con el alias que empleapor defecto Git.

```
REMOTE_ALIAS="origin"
```

1.1.6.2.3 *Estrategia de sincronización con repositorio remoto*

En los VCS distribuidos, como Git, aunque Fluho esté configurado para utilizar un repositorio remoto para centralizar los cambios en las versiones, es posible que los datos de este servidor remoto se lean de dos maneras (estrategias) diferentes:

- *Fetch*: Fluho lee los datos de una copia local de la base de datos del repositorio remoto.

  Esta estrategia permite que Fluho se ejecute mucho más rápido, pero los datos con los que trabaja pueden estar desactualizados si han sufrido cambios en el servidor desde la última vez que se recogieron de este.

  Si se emplea esta estrategia, se aconseja realizar una sincronización de la base de datos del repositorio local con la del remoto con cierta frecuencia. Esta sincronización se ha de realizar manualmente con el comando que provea a tal efecto el VCS.

  En Git, la base de datos del repositorio local se actualiza con la información del repositorio remoto mediante el comando `fetch` o `pull`, siendo el primero más recomendable ya que no actualiza el estado de los ficheros de la copia de trabajo.

- *Live*: Fluho lee los datos directamente del repositorio remoto.

  Esto permite tener siempre la información más actual, pero obliga a Fluho a interrogar al repositorio remoto cada vez que vaya a leer información sobre las versiones existentes, lo cual ralentiza notablemente el funcionamiento y puede incluso llegar a ser molesto si el servidor remoto solicita contraseña para cada acceso.

Por defecto, Fluho utiliza la estrategia '*fetch*'.

Esta configuración está establecida en el fichero `fluho.user` en la variable '`USE_REMOTEFETCH`':

```
USE_REMOTEFETCH=true
```

Para utilizar la estrategia '*live*', simplemente se debe establecer la variable '`USE_REMOTEFETCH`' a '*false*':

```
USE_REMOTEFETCH=false
```

Por otro lado, la escritura de los datos siempre se replica en el repositorio remoto directamente, independientemente de que se utilice la estrategia 'fetch' o 'live'.

En el caso de que se desee que los cambios sobre versiones no se repliquen en el repositorio remoto, se aconseja leer la sección de 'configuración' para deshabilitar el uso del repositorio remoto y la sección de 'acciones comunes' para forzar el trabajo sólo en local.

1.1.6.3 Mensajes mostrados antes de liberar una versión

Cuando el usuario ejecuta la liberación de una versión, puede recibir un mensaje para, por ejemplo, recordarle alguno puntos generales que debe cumplir antes de realizar la liberación.

Por ejemplo, al ir a liberar una versión para producción, por defecto se le muestra este mensaje:

```
Before releasing, make sure that the following points are met:
* Documentation has been updated
* Changes that require increasing minor or major number have not been introduced
* Code compiles correctly
* All tests have been passed
```

1.1.6.3.1 Para liberación de versión para pruebas

Editar el código fuente del fichero `fluho.user` y modificar el texto contenido en la función `show_releasefortesting_warning()`.

1.1.6.3.2 Para liberación de versión para producción

Editar el código fuente del fichero `fluho.user` y modificar el texto contenido en la función `show_releaseforproduction_warning()`.

1.1.6.4 Acciones extra al liberar una versión

Cuando se produce la liberación de una versión, Fluho ejecuta siempre las acciones elementales para reflejar esta en el VCS, pero se puede configurar para que además ejecute unas acciones extra particulares, como por ejemplo empaquetar el código fuente en un tarball y subirlo a un FTP o enviar una notificación a una dirección de email o cualquier otra cosa que se pueda necesitar.

Para facilitar la adición de esta programación, se han creado y organizado separadamente una serie de funciones que son llamadas siempre tras la ejecución exitosa de las acciones elementales. Eso sí, para rellenar estas se requiere tener conocimientos de programación.

1.1.6.4.1 Para liberación de versión para pruebas

Editar el código fuente del fichero `fluho.user` y modificar el contenido de la función

`do_releasefortesting_extra()`.

Además modificar las funciones `show_releasefortesting_extra_success()` y `show_releasefortesting_extra_error()` si se quieren mostrar mensajes personalizados para cuando estas acciones extras se lleven a cabo con éxito o con errores, respectivamente.

1.1.6.4.2 *Para liberación de versión para producción*

Editar el código fuente del fichero `fluho.user` y modificar el texto contenido en la función `show_releaseforproduction_warning()`.

Además modificar las funciones `show_releaseforproduction_extra_success()` y `show_releaseforproduction_extra_error()` si se quieren mostrar mensajes personalizados para cuando estas acciones extras se lleven a cabo con éxito o con errores, respectivamente.

1.1.7 Ejecución de acciones comunes

1.1.7.1 *Iniciar el control de versiones con fluho en un repositorio (init)*

```
$ fluho init
```

Nota: los siguientes comandos tendrían el mismo efecto:
```
$ fluho init v0.0
```
o
```
$ fluho checkout v0.0 --init
```

1.1.7.2 *Iniciar el desarrollo de una nueva versión (init)*

```
$ fluho init {major}.{minor}
```

Este comando deja listo el repositorio, tanto local como remoto si existe, para el desarrollo de la nueva versión, pero no cambia automáticamente a la rama de desarrollo de esta.

Para inicializar el desarrollo de la nueva versión y además cambiar a la rama de desarrollo de esta en un solo paso, utilizar el comando:

```
$ fluho checkout {major}.{minor} --init
```

1.1.7.3 *Mostrar las versiones existentes (list)*

```
$ fluho list
```
Se admiten los alias '`ls`' y '`version`', en lugar de '`list`'.
Esto mostrará un listado, ordenado de menor a mayor, de todas las versiones, tanto para desarrollo, como para pruebas y producción. La versión actual, es decir en la que se encuentre en ese momento, se mostrará marcada con un asterisco.

Para mostrar únicamente las versiones que tengan algunos de sus campos con un determinado valor, especificar estos campos dejando el resto en blanco. Ejemplos:

```
$ fluho list 1
```

```
* v1.0
  v1.0.0-rc
  v1.0.0-stable
  v1.1
  v1.2
  v1.3
```

```
  v1.4
  v1.5
  v1.6
  v1.7
```

```
$ fluho list ...-beta
```

```
  v0.0.9-beta
  v0.0.10-beta
  v0.0.11-beta
  v0.0.12-beta
```

Para mostrar únicamente las versiones existentes para un determinado propósito:

```
$ fluho list --purpose {purpose}
```

Donde `{purpose}` puede ser '`development`', '`testing`' o '`production`' o una abreviatura de cualquiera de ellos.

1.1.7.4 Mostrar la última versión existente (list)

```
$ fluho list --latest
```

Para mostrar la última versión que tenga algunos de sus campos con un determinado valor, especificar estos campos dejando el resto en blanco. Ejemplos:

```
$ fluho list 1 --latest
```

```
  v1.7
```

```
$ fluho list ...-beta --latest
```

```
  v0.0.12-beta
```

Para la última versión existente para un determinado propósito:

```
$ fluho list --latest --purpose {purpose}
```

Donde `{purpose}` puede ser '`development`', '`testing`' o '`production`' o una abreviatura de cualquiera de ellos.

1.1.7.5 Mostrar información sobre la versión actual (status)

```
$ fluho status
```

Se admite el alias '`st`', en lugar de '`status`'.

1.1.7.6 Cambiar a una versión (checkout)

1.1.7.6.1 Para desarrollo

```
$ fluho checkout {major}.{minor}
```

Se admite el alias '`co`', en lugar de '`checkout`'.

Aunque no es imprescindible, también es posible concretar que se desea cambiar a una versión de desarrollo, por ejemplo para forzar un error si se intenta ir a una versión de pruebas o de producción:

```
$ fluho checkout {major}.{minor} --purpose development
```

Nota: '`development`' se puede escribir parcialmente, por comodidad. Por ejemplo '`dev`' o '`d`'

Si lo que se desea es ir a la última versión *major*, *minor* de una *major*, se puede dejar en blanco el campo a averiguar y el programa sugerirá esta versión. Algunos ejemplos serían:

```
$ fluho checkout --purpose development
```

```
error: Version not completely specified as {major}.{minor}
To checkout to the latest version for development, use:

    $ fluho checkout v30.0
```

```
$ fluho checkout 1 --purpose development
```

```
error: Version not completely specified as {major}.{minor}
To checkout to the latest version for development matching the given data, use:

    $ fluho checkout v1.7
```

1.1.7.6.2 *Para pruebas*

```
$ fluho checkout {major}.{minor}.{rev}-{status}
```

Aunque no es imprescindible, también es posible concretar que se desea cambiar a una versión de pruebas, por ejemplo para forzar un error si se intenta ir a una versión de desarrollo o de producción:

```
$ fluho checkout {major}.{minor}.{rev}-{status} --purpose testing
```

Nota: '`testing`' se puede escribir parcialmente, por comodidad. Por ejemplo '`test`' o '`t`'

Si lo que se desea es ir a la última versión *major*, *minor* de una *major*, revisión o estado, se puede dejar en blanco el campo a averiguar y el programa sugerirá esta versión. Algunos ejemplos serían:

```
$ fluho checkout --purpose testing
```

```
error: Version not completely specified as {major}.{minor}.{rev}-{status}
To checkout to the latest version for testing, use:

    $ fluho checkout v1.0.0-rc
```

```
$ fluho checkout 0.1 --purpose testing
```

```
error: Version not completely specified as {major}.{minor}.{rev}-{status}
To checkout to the latest version for testing matching the given data, use:

    $ fluho checkout v0.1.1-alpha
```

1.1.7.6.3 *Para producción*

```
$ fluho checkout {major}.{minor}.{rev}-stable
```

Nota: '`-stable`' se puede omitir

Aunque no es imprescindible, también es posible concretar que se desea cambiar a una versión de pruebas, por ejemplo para forzar un error si se intenta ir a una versión de desarrollo o de producción:

```
$ fluho checkout {major}.{minor}.{rev}-stable --purpose production
```

Nota: '`production`' se puede escribir parcialmente, por comodidad. Por ejemplo '`prod`' o '`p`'

Si lo que se desea es ir a la última versión *major*, *minor* de una *major*, revisión o estado, se puede dejar en blanco el campo a averiguar y el programa sugerirá esta versión. Algunos ejemplos serían:

```
$ fluho checkout --purpose production
```

```
error: Version not completely specified as {major}.{minor}.{rev}-stable
To checkout to the latest version for production, use:

    $ fluho checkout v1.0.0-stable
```

```
$ fluho checkout 0 --purpose production
```

```
error: Version not completely specified as {major}.{minor}.{rev}-stable
To checkout to the latest version for production matching the given data, use:

    $ fluho checkout v0.0.21-stable
```

1.1.7.7 *Desarrollar una funcionalidad de una versión*

Esta funcionalidad es opcional. Todos los desarrollos se pueden llevar a cabo sobre la versión major.minor en general, pero si se desea separar el desarrollo de una funcionalidad para posteriormente fundir el código de esta en la versión major.minor general, seguir este procedimiento:

1. Asegurarse de que se está en el estadio de desarrollo de la versión major.minor deseada:
   `$ fluho checkout {major}.{minor}`
2. Iniciar el desarrollo de la funcionalidad:
   `$ fluho init --feature {nombre_funcionalidad}`
3. Cambiar al desarrollo de la funcionalidad:
   `$ fluho checkout {major}.{minor} --feature {nombre_funcionalidad}`
4. Realizar el desarrollo de la funcionalidad.
5. Terminar el desarrollo de la funcionalidad:
   `$ fluho release {major}.{minor} --feature {nombre_funcionalidad}`

Nota: los pasos del 1 al 3 se pueden unificar en uno solo:

```
$ fluho checkout {major}.{minor} --feature {nombre_funcionalidad} --init
```

1.1.7.8 *Liberar una versión (release)*

```
$ fluho release {version}
```

Donde `{version}` debe especificarse el número de versión en el formato {*major*}.{*minor*}.{*rev*}-{*status*}. Se admite el alias '`re`', en lugar de '`release`'.

Si se desconoce el siguiente número de revisión que correspondería liberar, dejar `{version}` en blanco y el programa lo sugerirá.

La aplicación sólo permitirá liberar una versión de pruebas (*alpha*, *beta* o rc*)* desde el estadio de desarrollo correspondiente, y una versión de producción (*stable*) desde el estadio de pruebas de la misma revisión con *status rc*. Tampoco permitirá retroceder en el *status* de una versión *major*.*minor*, excepto de *stable* a *rc*.

Se mostrará un mensaje recordando las comprobaciones que deben estar satisfechas antes de liberar una versión de pruebas o de producción y a continuación se solicitará confirmación interactiva.

1.1.7.9 *Cambiar el repositorio remoto (--remote)*

Se realiza con el objetivo de especificar un alias del repositorio remoto, distinto al de por defecto (*origin* en Git), donde se encuentran centralizadas las versiones creadas y liberadas.

Esta modificación se puede llevar a cabo de forma global, editando la configuración del programa, o de forma limitada a un comando, mediante un argumento.

Para hacerlo de forma global, se puede encontrar los pasos necesarios en la sección de configuración.

Para que el cambio sólo afecte a una ejecución en concreto, añadir el argumento `--remote {remote_alias}` a la ejecución de cualquier otra acción. Por ejemplo:

```
$ fluho init v1.0 --remote mycentralrepo
```

En el caso de que se desee trabajar habitualmente con un repositorio remoto distinto al de la configuración del programa, por comodidad se aconseja crear un alias que siempre apunte al repositorio remoto deseado. Por ejemplo:

```
$ alias fluho_prj='fluho --remote mycentralrepo'
```

1.1.7.10 *Forzar trabajar solo en local (--noremote)*

Si no hay un repositorio remoto configurado en VCS, Fluho trabajará automáticamente en local. En el caso de que sí exista un repositorio remoto configurado en el VCS y la variable '`USE_REMOTE`' de Fluho esté establecida a '*true*', todas las acciones sobre las versiones se compartirán en este.

Esta replicación de las acciones sobre versiones en el repositorio remoto se puede desactivar de forma global, editando la configuración del programa, o de forma limitada a un comando, mediante un argumento.

Para hacerlo de forma global, consultar la sección de configuración.

Para que el cambio sólo afecte a una ejecución en concreto, añadir el argumento `--noremote` a la ejecución de cualquier otra acción. Por ejemplo:

```
$ fluho init v1.0 --noremote
```

En el caso de que se desee trabajar habitualmente sin un repositorio remoto pero sin desactivar esta funcionalidad de forma global, por comodidad se aconseja crear un alias. Por ejemplo:

```
$ alias fluho_nr='fluho --noremote'
```

1.1.7.11 *Replicar versiones locales en un repositorio remoto (sync)*

Si se ha trabajado con Fluho solamente en local y, llegado el momento, se desea compartir las versiones locales en un repositorio remoto, ejecutar:

```
$ fluho sync
```

Nota: asegurarse de que la variable '`USE_REMOTE`' esté establecida a '*false*' en la configuración general de Fluho o que se añade el argumento `--remote {remote_alias}` al comando.

1.1.8 Ejemplo de acciones ejecutadas sobre repositorio

Evento	Acciones

Comienzo de desarrollo
 	
`$ fluho init`
o
`$ fluho init 0`
o
`$ fluho init 0.0`
	6. Creación de etiqueta *v0.0.0-alpha* ◄
	7. Creación de etiqueta *v0.0-alpha1*
	8. Creación de etiqueta *v0.0.0*
	9. Creación de rama *test/cutting-edge* a partir de rama *cutting-edge* (*master*)
	10. Creación de rama *v0* a partir de rama *cutting-edge*
	11. Creación de rama *test/v0* a partir de rama *cutting-edge*
	12. Creación de rama *v0.0* a partir de rama *v0*
	13. Creación de rama *test/v0.0* a partir de rama *v0*
	14. Creación de rama *dev/v0.0* a partir de rama *v0* ◄
	15. Cambio a rama *dev/v0.0* (en local) ◄

Comienzo de versión nueva `0.1`	1. Creación de rama *v0.1* a partir de la rama *v0*

(incremento en *minor*) A. `$ fluho init 0.1` B. ---	2. Creación de rama *test/v0.1* a partir de la rama *v0* 3. Creación de rama *dev/v0.1* a partir de la rama *v0* ◄
Selección de versión `0.1` para desarrollo A. `$ fluho checkout 0.1` B. `$ fluho checkout 0.1 --init`	1. Cambio a rama *dev/v0.1* (en local) ◄
Desarrollo sobre versión `0.1` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v0.1* ◄
Liberación de versión `0.1.0-alpha` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v0.1.0-alpha`	1. Creación de etiqueta *v0.1.0-alpha* ◄ 2. Creación de etiqueta *v0.1-alpha1* 3. Fusión de rama *dev/v0.1* en rama *test/v0.1* con *ff* 4. Fusión de rama *test/v0.1* en rama *test/v0* con *ff* 5. Fusión de rama *test/v0* en rama *test/cutting-edge* con *ff*
Comienzo de versión nueva `1.0` (incremento en *major*) A. `$ fluho init 1.0` B. ---	1. Creación de rama *v1* a partir de la rama *cutting-edge* 2. Creación de rama *test/v1* a partir de la rama *v1* 3. Creación de rama *v1.0* a partir de la rama *v1* 4. Creación de rama *test/v1.0* a partir de la rama *v1.0* 5. Creación de rama *dev/v1.0* a partir de la rama *test/v1.0* ◄
Selección de versión `1.0` para desarrollo A. `$ fluho checkout 1.0` B. `$ fluho checkout 1.0 --init`	1. Cambio a rama *dev/v1.0* (en local) ◄
Desarrollo sobre versión `1.0` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.0* ◄
Liberación de versión `1.0.0-alpha` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.0-alpha`	1. Creación de etiqueta *v1.0.0-alpha* ◄ 2. Creación de etiqueta *v1.0-alpha1* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff* 4. Fusión de rama *test/v1.0* en rama *test/v1* con *ff* 5. Fusión de rama *test/v1* en rama *test/cutting-edge* con *ff*
Liberación de versión `1.0.1-alpha` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.1-alpha`	1. Creación de etiqueta *v1.0.1-alpha* ◄ 2. Creación de etiqueta *v1.0-alpha2* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff* 4. Fusión de rama *test/v1.0* en rama *test/v1* con *ff* 5. Fusión de rama *test/v1* en rama *test/cutting-edge* con *ff*
Liberación de versión `1.0.2-beta` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.2-beta`	1. Creación de etiqueta *v1.0.2-beta* ◄ 2. Creación de etiqueta *v1.0-beta1* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff* 4. Fusión de rama *test/v1.0* en rama *test/v1* con *ff* 5. Fusión de rama *test/v1* en rama *test/cutting-edge* con *ff*
Liberación de versión `1.0.3-rc` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.3-rc`	1. Creación de etiqueta *v1.0.3-rc* ◄ 2. Creación de etiqueta *v1.0-rc1* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff* 4. Fusión de rama *test/v1.0* en rama *test/v1* con *ff* 5. Fusión de rama *test/v1* en rama *test/cutting-edge* con *ff*
Selección de la versión `1.0.3-rc` para testing `$ fluho checkout v1.0.3-rc`	1. Cambio a la etiqueta *v1.0.3-rc* (en local) ◄
Liberación de versión `1.0.3-stable` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.3-stable`	1. Creación de etiqueta *v1.0.3-stable* ◄ 2. Creación de etiqueta *v1.0-stable1* 3. Creación de etiqueta *v1.0.3* 4. Fusión de rama *test/v1.0* en rama *v1.0* con *ff* 5. Fusión de rama *v1.0* en rama *v1* con *ff* 6. Fusión de rama *v1* en rama *cutting-edge* con *ff*
Paso al estadio de desarrollo de la versión `1.0` `$ fluho checkout v1.0`	1. Cambio a rama *dev/v1.0* (en local) ◄
Liberación de versión `1.0.4-rc`	1. Creación de etiqueta *v1.0.4-rc* ◄

(tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.4-rc`	2. Creación de etiqueta *v1.0-rc2* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff* 4. Fusión de rama *test/v1.0* en rama *test/v1* con *ff* 5. Fusión de rama *test/v1* en rama *test/cutting-edge* con *ff*
Selección de la versión `1.0.4-rc` para testing `$ fluho checkout v1.0.4-rc`	1. Cambio a la etiqueta *v1.0.4-rc* (en local) ◄
Liberación de versión `1.0.4-stable` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.0.4-stable`	1. Creación de etiqueta *v1.0.4-stable* ◄ 2. Creación de etiqueta *v1.0-stable2* 3. Creación de etiqueta *v1.0.4* 4. Fusión de rama *test/v1.0* en rama *v1.0* con *ff* 5. Fusión de rama *v1.0* en rama *v1* con *ff* 6. Fusión de rama *v1* en rama *cutting-edge* con *ff*
Comienzo de versión nueva `1.1` (incremento en *minor*) A. `$ fluho init 1.1` B. ---	1. Creación de rama *v1.1* a partir de la rama *v1* 2. Creación de rama *test/v1.1* a partir de la rama *v1* 3. Creación de rama *dev/v1.1* a partir de la rama *v1* ◄
Selección de versión `1.1` para desarrollo A. `$ fluho checkout v1.1` B. `$ fluho checkout v1.1 --init`	1. Cambio a rama *dev/v1.1* (en local) ◄
Desarrollo sobre versión `1.1` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.1* ◄
Liberación de versión `1.1.0-rc` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.1.0-rc`	1. Creación de etiqueta *v1.1.0-rc* ◄ 2. Creación de etiqueta *v1.1-rc1* 3. Fusión de rama *dev/v1.1* en rama *test/v1.1* con *ff* 4. Fusión de rama *test/v1.1* en rama *test/v1* con *ff* 5. Fusión de rama *test/v1* en rama *test/cutting-edge* con *ff*
Selección de la versión `1.1.0-rc` para testing `$ fluho checkout v1.1.0-rc`	1. Cambio a la etiqueta *v1.1.0-rc* (en local) ◄
Liberación de versión `1.1.0-stable` (primera estable) (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v1.1.0-stable`	1. Creación de etiqueta *v1.1.0-stable* ◄ 2. Creación de etiqueta *v1.1-stable1* 3. Creación de etiqueta *v1.1.0* 4. Fusión de rama *test/v1.1* en rama *v1.1* con ff 5. Fusión de rama *v1.1* en rama *v1* con *ff* 6. Fusión de rama *v1* en rama *cutting-edge* con *ff*
Selección de versión `1.0` para desarrollo `$ fluho checkout v1.0`	1. Cambio a rama *dev/v1.0* (en local) ◄
Desarrollo sobre versión `1.0` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.0* ◄
Liberación de versión `1.0.5-rc` (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release v1.0.5-rc`	1. Creación de etiqueta *v1.0.5-rc* ◄ 2. Creación de etiqueta *v1.0-rc3* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff*
Selección de la versión `1.0.5-rc` para testing `$ fluho checkout v1.0.5-rc`	1. Cambio a la etiqueta *v1.0.5-rc* (en local) ◄
Liberación de versión `1.0.5-stable` (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release v1.0.5-stable`	1. Creación de etiqueta *v1.0.5-stable* ◄ 2. Creación de etiqueta *v1.0-stable3* 3. Creación de etiqueta *v1.0.5* 4. Fusión de rama *test/v1.0* en rama *v1.0* con *ff*
Comienzo de versión nueva `v2.0` (incremento en *major*) A. `$ fluho init v2` B. --- Nota: Obsérvese que se puede omitir el *minor* si es cero.	1. Creación de rama *v2* a partir de la rama *cutting-edge* 2. Creación de rama *test/v2* a partir de la rama *v2* 3. Creación de rama *v2.0* a partir de la rama *v2* 4. Creación de rama *test/v2.0* a partir de la rama *v2.0* 5. Creación de rama *dev/v2.0* a partir de la rama *test/v2.0* ◄
Selección de versión `2.0` para desarrollo A. `$ fluho checkout v2.0`	1. Cambio a rama *dev/v2.0* (en local) ◄

B. `$ fluho checkout v2 --init`	
Desarrollo sobre versión `2.0` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v2.0* ◄
Liberación de versión `2.0.0-alpha` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v2.0.0-alpha`	1. Creación de etiqueta *v2.0.0-alpha* ◄ 2. Creación de etiqueta *v2.0-alpha1* 3. Fusión de rama *dev/v2.0* en rama *test/v2.0* con *ff* 4. Fusión de rama *test/v2.0* en rama *test/v2* con *ff* 5. Fusión de rama *test/v2* en rama *test/cutting-edge* con *ff*
Selección de versión `1.0` para desarrollo `$ fluho checkout v1.0`	1. Cambio a rama *dev/v1.0* (en local) ◄
Desarrollo sobre versión `1.0` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.0* ◄
Liberación de versión `1.0.6-rc` (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release v1.0.6-rc`	1. Creación de etiqueta *v1.0.6-rc* ◄ 2. Creación de etiqueta *v1.0-rc4* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff*
Liberación de versión `1.0.7-rc` (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release v1.0.7-rc`	1. Creación de etiqueta *v1.0.7-rc* ◄ 2. Creación de etiqueta *v1.0-rc5* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff*
Selección de la versión `1.0.7-rc` para testing `$ fluho checkout v1.0.7-rc`	1. Cambio a la etiqueta *v1.0.7-rc* (en local) ◄
Liberación de versión `1.0.7-stable` (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release 1.0.7-stable`	1. Creación de etiqueta *v1.0.7-stable* ◄ 2. Creación de etiqueta *v1.0-stable4* 3. Creación de etiqueta *v1.0.7* 4. Fusión de rama *test/v1.0* en rama *v1.0* con *ff*
Selección de versión `1.1` para desarrollo `$ fluho checkout 1` Nota: Al ser 1.1 *head* en *minor*, es posible omitir el *minor*	1. Cambio a rama *dev/v1.1* (en local) ◄
Desarrollo sobre versión `1.1` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.1* ◄
Liberación de versión `1.1.1-rc` (tipo *maintenance-1*: es *head* en *minor* pero no en *major*) `$ fluho release v1.1.0-rc`	1. Creación de etiqueta *v1.1.1-rc* ◄ 2. Creación de etiqueta *v1.1-rc2* 3. Fusión de rama *dev/v1.1* en rama *test/v1.1* con *ff* 4. Fusión de rama *test/v1.1* en rama *test/v1* con *ff*
Selección de la versión `1.1.1-rc` para testing `$ fluho checkout v1.1.1-rc`	1. Cambio a la etiqueta *v1.1.1-rc* (en local) ◄
Liberación de versión `1.1.1-stable` (tipo *maintenance-1*: es *head* en *minor* pero no en *major*) `$ fluho release 1.1.1-stable`	1. Creación de etiqueta *v1.1.1-stable* ◄ 2. Creación de etiqueta *v1.1-stable2* 3. Creación de etiqueta *v1.1.1* 4. Fusión de rama *test/v1.1* en rama *v1.1* con *ff* 5. Fusión de rama *v1.1* en rama *v1* con *ff*
Comienzo de versión nueva `1.2` (incremento en *minor*) A. `$ fluho init 1.2` B. ---	1. Creación de rama *v1.2* a partir de la rama *v1* 2. Creación de rama *test/v1.2* a partir de la rama *v1* 3. Creación de rama *dev/v1.2* a partir de la rama *v1* ◄
Selección de versión `1.2` para desarrollo A. `$ fluho checkout 1.2` B. `$ fluho checkout 1.2 --init`	1. Cambio a rama *dev/v1.2* (en local) ◄
Desarrollo sobre versión `1.2` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.2* ◄
Liberación de versión `1.2.0-rc` (tipo *maintenance-1*: es *head* en *minor* pero no en *major*) `$ fluho release 1.2.0-rc`	1. Creación de etiqueta *v1.2.0-rc* ◄ 2. Creación de etiqueta *v1.2-rc1* 3. Fusión de rama *dev/v1.2* en rama *test/v1.2* con *ff* 4. Fusión de rama *test/v1.2* en rama *test/v1* con *ff*
Selección de la versión `1.2.0-rc` para testing `$ fluho checkout v1.2.0-rc`	1. Cambio a la etiqueta *v1.2.0-rc* (en local) ◄

Liberación de versión `1.2.0-stable` (tipo *maintenance-1*: es *head* en *minor* pero no en *major*) `$ fluho release 1.2.0-stable`	1. Creación de etiqueta *v1.2.0-stable* ◄ 2. Creación de etiqueta *v1.2-stable1* 3. Creación de etiqueta *v1.2.0* 4. Fusión de etiqueta *1.2.0-rc* en rama *v1.2* con *ff* 5. Fusión de rama *v1.2* en rama *v1* con *ff*
Selección de versión `1.2` para desarrollo `$ fluho checkout 1.2`	1. Cambio a rama *dev/v1.2* (en local) ◄
~~Liberación de versión `1.2.1-beta`~~ ~~(tipo *maintenance-1*: es *head* en *minor* pero no en *major*)~~ ~~`$ fluho release 1.2.1-beta`~~ Nota: no admitido retroceder estado de la versión a *beta*	1. ~~Creación de etiqueta *v1.2.1-beta* ◄~~ 2. ~~Creación de etiqueta *v1.2-beta1*~~ 3. ~~Fusión de rama *dev/v1.2* en rama *test/v1.2* con *ff*~~ 4. ~~Fusión de rama *test/v1.2* en rama *test/v1* con *ff*~~
Selección de versión `2.0` para desarrollo `$ fluho checkout 2.0`	1. Cambio a rama *dev/v2.0* (en local) ◄
Liberación de versión `2.0.1-beta` (tipo *cutting-edge*: es *head* en *major* y en *minor*) `$ fluho release v2.0.1-beta`	1. Creación de etiqueta *v2.0.1-beta* ◄ 2. Creación de etiqueta *v2.0-beta1* 3. Fusión de rama *dev/v2.0* en rama *test/v2.0* con *ff* 4. Fusión de rama *test/v2.0* en rama *test/v2* con *ff*
Selección de la versión `2.0.1-beta` para testing `$ fluho checkout v2.0.1-beta`	1. Cambio a la etiqueta *v2.0.1-beta* (en local) ◄
~~Liberación de versión `2.0.1-stable`~~ ~~(tipo *cutting-edge*: es *head* en *major* y en *minor*)~~ ~~`$ fluho release v2.0.1-stable`~~ Nota: no se puede liberar una versión *stable* desde una *beta*, debe realizarse siempre desde una *rc*.	1. ~~Creación de etiqueta *v2.0.1-stable* ◄~~ 2. ~~Creación de etiqueta *v2.0.1-stable1*~~ 3. ~~Creación de etiqueta *v2.0.1*~~ 4. ~~Fusión de etiqueta *2.0.1-beta* en rama *v2.0* con *ff*~~ 5. ~~Fusión de rama *v2.0* en rama *v2* con *ff*~~ 6. ~~Fusión de rama *v2* en rama *cutting-edge* con *ff*~~
Selección de la última versión `1.2` para producción `$ fluho checkout 1.2 --purpose production`	1. Cambio a rama *v1.2* (en local) ◄
~~Intento de desarrollo sobre versión 1.2 para producción~~ Nota: no se pueden subir cambios en rama de producción	1. ~~Subida de paquetes de cambios nuevos a rama *v1.2*~~

Selección de la última versión `1.2` para pruebas
`$ fluho checkout 1.2 --purpose testing`
Obtención de la sugerencia de última versión
`$ fluho checkout 1.2.0-rc`
	1. Cambio a la etiqueta *v1.2.0-rc* (en local) ◄

~~Intento de desarrollo sobre versión 1.2 para pruebas~~ Nota: no se pueden subir cambios en rama de pruebas	1. ~~Subida de paquetes de cambios nuevos a rama *v1.2*~~
Selección de versión 1.0.3-stable `$ fluho checkout 1.0.3-stable`	1. Cambio a etiqueta v1.0.3-stable (en local) ◄
~~Intento de desarrollo sobre versión 1.0.3-stable~~ Nota: no se pueden subir cambios a partir de una revisión.	1. ~~Subida de paquetes de cambios nuevos a partir de la etiqueta v1.0.3-stable~~

Selección de versión 1.0.2
`$ fluho checkout 1.0.2`
Obtención de la sugerencia del estado de esta revisión
`$ fluho checkout 1.0.2-beta`
	1. Cambio a etiqueta v1.0.2-beta (en local) ◄

~~Intento de desarrollo sobre versión 1.0.2-beta~~ Nota: no se pueden subir cambios a partir de una revisión.	1. ~~Subida de paquetes de cambios nuevos a partir de la etiqueta v1.0.2-beta~~
~~Selección de versión 1.0.8-stable para desarrollo~~ ~~`$ fluho checkout 1.0.8-stable`~~ Nota: La versión aún no existe porque no ha sido liberada	~~1. Cambio a etiqueta v1.0.8-stable (en local) ◄~~
~~Selección de versión 3.0 para desarrollo~~ ~~`$ fluho checkout 3.0`~~ Nota: La versión aún no existe porque no ha sido creada	~~1. Cambio a rama *dev/v3.0* (en local) ◄~~
Selección de versión `1.0` para desarrollo `$ fluho checkout v1.0`	1. Cambio a rama *dev/v1.0* (en local) ◄

Desarrollo sobre versión `1.0` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v1.0* ◄
Liberación de versión `1.0.8-rc` (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release v1.0.8-rc`	1. Creación de etiqueta *v1.0.8-rc* ◄ 2. Creación de etiqueta *v1.0-rc6* 3. Fusión de rama *dev/v1.0* en rama *test/v1.0* con *ff*
Selección de la versión `1.0.8-rc` para testing `$ fluho checkout v1.0.8-rc`	1. Cambio a la etiqueta *v1.0.8-rc* (en local) ◄
Liberación de versión `1.0.8-stable` (revisión de estable) (tipo *maintenance-2*: no es *head* en *minor*) `$ fluho release 1.0.8-stable`	1. Creación de etiqueta *v1.0.8-stable* ◄ 2. Creación de etiqueta *v1.0-stable5* 3. Creación de etiqueta *v1.0.8* 4. Fusión de rama *dev/v1.0* en rama *v1.0* con *ff*
Comienzo de versión nueva `2.5` (incremento en *minor* no consecutivo) `$ fluho init 2.5`	1. Creación de rama *v2.5* a partir de la rama *v2* 2. Creación de rama *test/v2.5* a partir de la rama *v2* 3. Creación de rama *dev/v2.5* a partir de la rama *v2* ◄
Comienzo de versión nueva `v2016.0` (incremento en *major* no consecutivo) `$ fluho init v2016.0`	1. Creación de rama *v2016* a partir de la rama *cutting-edge* 2. Creación de rama *test/v2016* a partir de la rama *v2016* 3. Creación de rama *v2016.0* a partir de la rama *v2016* 4. Creación de rama *test/v2016.0* a partir de rama *v2016.0* 5. Creación de rama *dev/v2016.0* a partir de rama *test/v2016.0* ◄
Comienzo de versión nueva `v2017.4` (incremento en *major* no consecutivo) `$ fluho init v2017.4`	1. Creación de rama *v2017* a partir de la rama *cutting-edge* 2. Creación de rama *test/v2017* a partir de la rama *v2017* 3. Creación de rama *v2017.4* a partir de la rama *v2017* 4. Creación de rama *test/v2017.4* a partir de rama *v2017.4* 5. Creación de rama *dev/v2017.4* a partir de rama *test/v2017.4* ◄
Comienzo de versión nueva `v2015.0` (decremento en *major*) `$ fluho init v2015.0`	1. Creación de rama *v2015* a partir de la rama *v2* 2. Creación de rama *test/v2015* a partir de la rama *v2015* 3. Creación de rama *v2015.0* a partir de la rama *v2015* 4. Creación de rama *test/v2015.0* a partir de rama *v2015.0* 5. Creación de rama *dev/v2015.0* a partir de rama *test/v2015.0* ◄
Comienzo de nueva *feature* `foo` de la versión `v1.2` A. `$ fluho init v1.2 --feature foo` B. ---	1. Creación de rama *dev/v1.2/foo* a partir de la rama *dev/v1.2* ◄
Selección de *feature* `foo` de versión `v2015` para desarrollo A. `$ fluho checkout v1.2 --feature foo` B. `$ fluho checkout v1.2 --feature foo --init`	1. Cambio a rama *dev/v1.2/foo* (en local) ◄
Desarrollo sobre *feature* `foo` de la versión `v1.2`	1. Subida de paquetes de cambios a rama *dev/v1.2/foo* ◄
Liberación de la *feature* `foo` de la version `v1.2` `$ fluho release v1.2 --feature foo`	1. Fusión de rama *dev/v1.2/foo* en rama *dev/v1.2* sin ff ◄ 2. Eliminación de rama *dev/v1.2/foo* ◄

Notas:

- *Todas las acciones se refieren a ser realizadas sobre el repositorio remoto (normalmente primero en la working copy local y después sincronizando con el repositorio remoto), salvo que se especifique en alguna que esta debe ser realizada únicamente en local.*
- *Aunque no se especifique, entre la liberación de una versión y la de otra, normalmente existirá la subida de paquetes de cambios:*

Desarrollo sobre versión `{major}.{minor}` para desarrollo	1. Subida de paquetes de cambios nuevos a rama *dev/v{major}.{minor}* ◄

1.1.9 Troubleshooting

1.1.9.1 *Commits subidos por error a una rama NO de desarrollo*

IMPORTANTE: Utilizar únicamente este procedimiento cuando los commits a eliminar sean los últimos de la rama.

1. Salvar los ficheros con cambios subidos a la rama erronea.
2. Habilitar *non-fastforward* en el repositorio remoto

   Desde la máquina local, ejecutar:

   ```
   $ ssh {remote_user}@{remote_repo_ip} nano {git_repo_path}/config
   ```

   Cambiar la línea:

   ```
           denyNonFastforwards = true
   ```

   Por:

   ```
           denyNonFastforwards = false
   ```

   Salvar los cambios.
3. `$ git checkout {rama_con_commits_erroneos}`
4. `$ git reset --hard {hash_ultimo_commit_correcto}`
5. `$ git push --force`
6. Vovler a deshabilitar *non-fastforward* en el repositorio remoto.

   Desde la máquina local, ejecutar:

   ```
   $ ssh {remote_user}@{remote_repo_ip} nano {git_repo_path}/config
   ```

   Cambiar la línea:

   ```
           denyNonFastforwards = false
   ```

   Por:

   ```
           denyNonFastforwards = true
   ```

   Salvar los cambios.

Plantilla v20160825

2. Heading 1

2.1 Heading 2

2.1.1 Heading 3

2.1.1.1 *Heading 4*

2.1.1.1.1 *Heading 5*

2.1.1.1.1.1 **Heading 6**

Text body

`Source Text`

```
Source Text Boxed
```

```
Source Text Small
```

```
Source Text Small Boxed
```

Término Técnico

Aclaración

`Cita`

Fórmula

Internal link

`Character - Source Text`

`Filename`

Pregunta

Nota

ToDo

Table Heading	Table Heading
Table Content	Table Content

- Lista Principal 1
  Lista Principal 1

- Lista Principal 2
  Lista Principal 2
  - Lista Principal 3
    Lista Principal 3
    - Lista Principal 4
      Lista Principal 4
      - Lista Principal 5
        Lista Principal 5

- Lista Secundaria 1
  Lista Secundaria 1
  - Lista Secundaria 2
    Lista Secundaria 2
    - Lista Secundaria 3
      Lista Secundaria 3
      - Lista Secundaria 4
        Lista Secundaria 4
        - Lista Secundaria 5
          Lista Secundaria 5

- Lista sencilla 1
  Lista sencilla 1
  - Lista sencilla 2
    Lista sencilla 2
    - Lista sencilla 3
      Lista sencilla 3
      - Lista sencilla 4
        Lista sencilla 4
        - Lista sencilla 5
          Lista sencilla 5

2. Lista Numerada Sencilla 1
   Lista Numerada Sencilla 1
   1. Lista Numerada Sencilla 2
      Lista Numerada Sencilla 2
      1. Lista Numerada Sencilla 3
         Lista Numerada Sencilla 3
         1. Lista Numerada Sencilla 4
            Lista Numerada Sencilla 4
            1. Lista Numerada Sencilla 5
               Lista Numerada Sencilla 5

7. Lista Numerada Niveles 1
   Lista Numerada Niveles 1

7.1. Lista Numerada Niveles 2
Lista Numerada Niveles 2

7.1.1. Lista Numerada Niveles 3
Lista Numerada Niveles 3

7.1.1.1. Lista Numerada Niveles 4
Lista Numerada Niveles 4

7.1.1.1.1. Lista Numerada Niveles 5
Lista Numerada Niveles 5