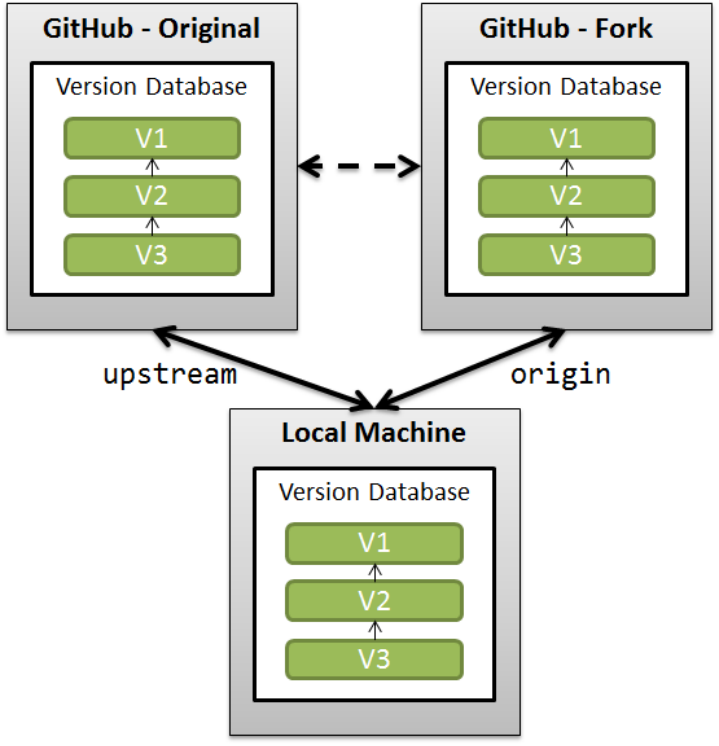
Taller de introducción a GIT

SCM

Tarea 01: Copiar repositorios

- Ingresar a "github.com" con un usuario y contraseña
- Ingresar a https://github.com/mart-dominguez/taller-git-02 y seleccionar la opción **"fork"**
- ¿Qué hace "fork"? Crea una copia exacta del repositorio, pero no en nuestro espacio local, sino en otro espacio del repositorio remoto.

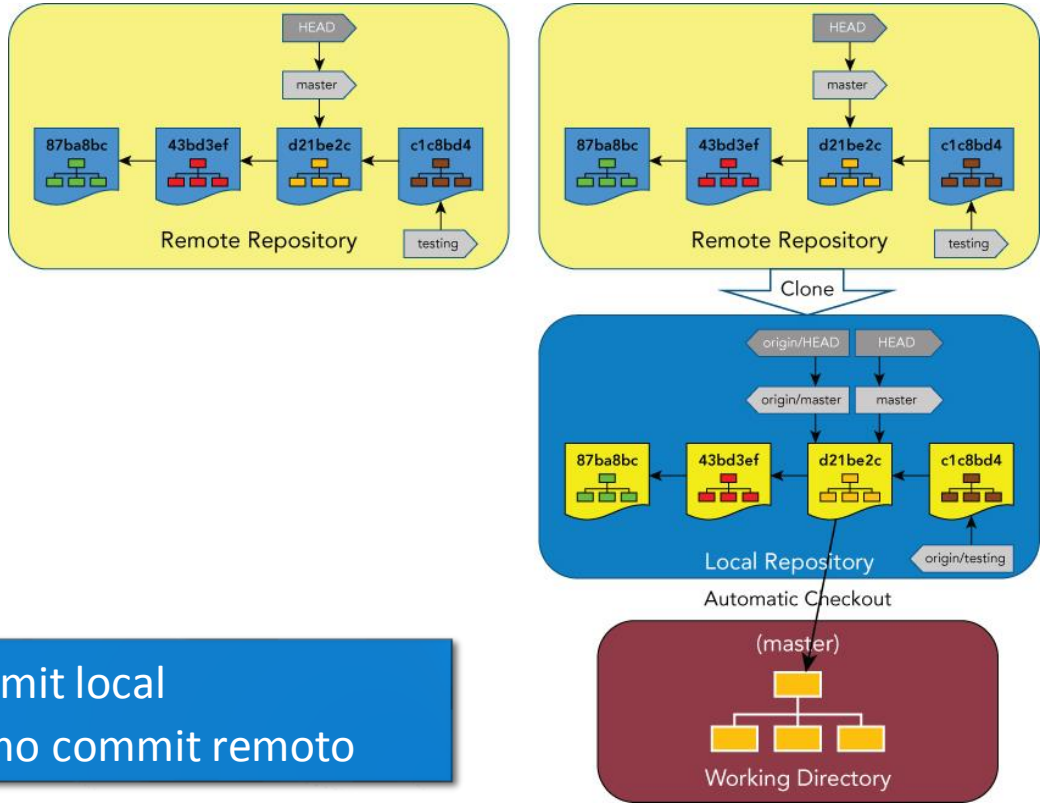


Tarea 02: clonar repositorios

- Git no está permanentemente conectado al repositorio remoto.
- Solo cuando tiene que interactuar con el mismo intercambia información de estado para realizar correctamente las operaciones requeridas.
  - Mediante **pull** o **push**, para descargar los commits que se necesitan.
- Una de las formas de iniciar el trabajo en git es “clonar” un repositorio remoto.
  - **$ git clone** **http://gitserver/repository**
- Cuando realizamos un clone ocurren varias cosas
  - Se copia el directorio .git del servidor remoto
  - Git crea todas las ramas que figuran en el repositorio remoto en el local.
  - Luego realiza un checkout del HEAD actual del master.

Clonar repositorios

- Al clonar un repositorio, descargamos toda la base de datos con la historia del repositorio (tags, branches, commits)
- Localmente se guardan 2 punteros



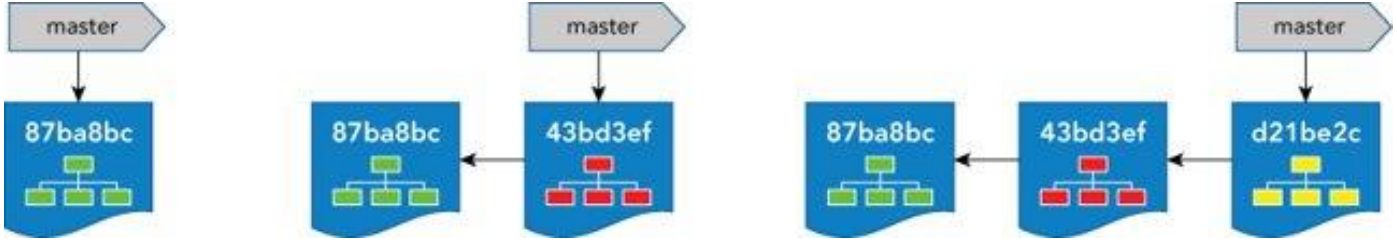
- HEAD: ultimo commit local
- Origin/HEAD: ultimo commit remoto

Ramas

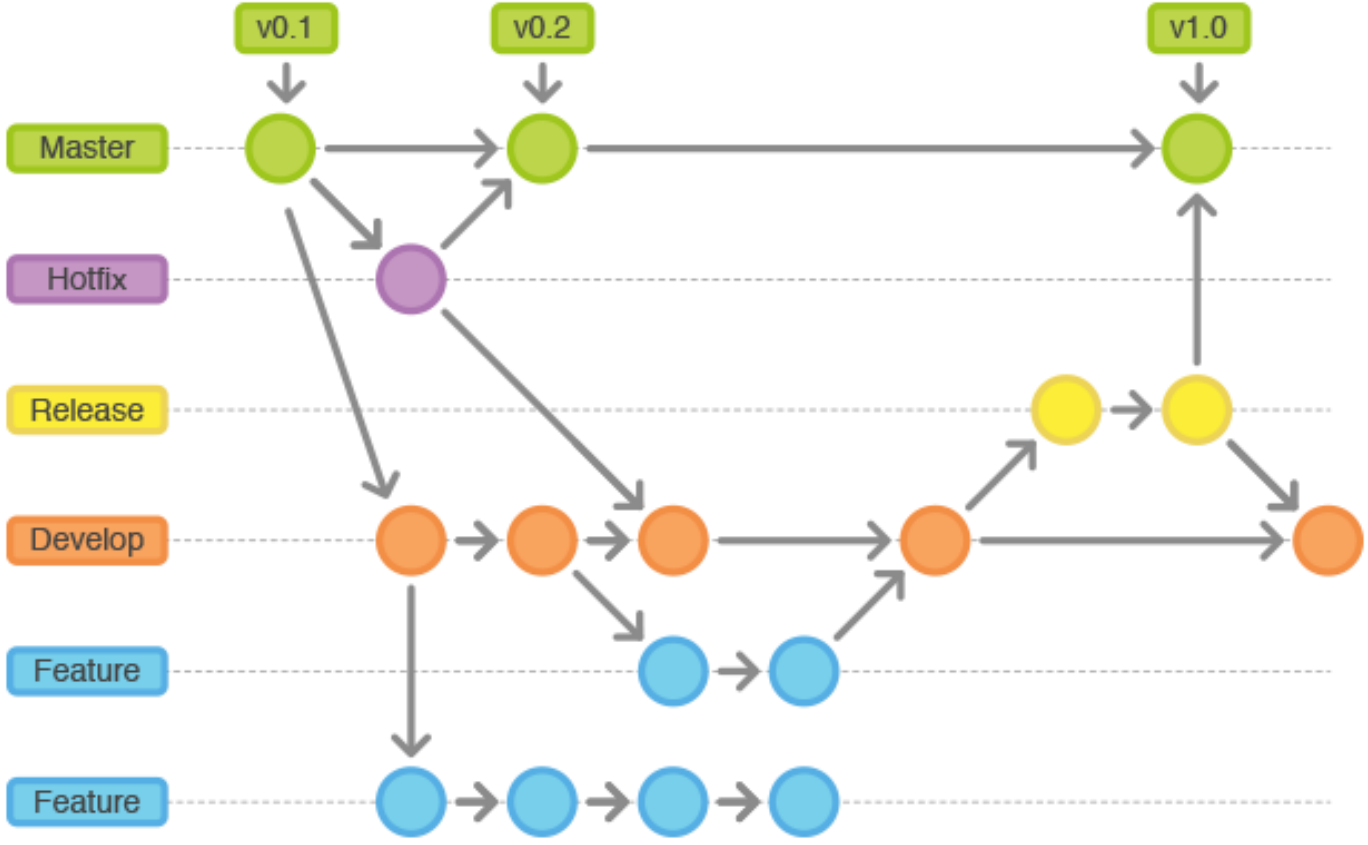
- Ramas: división de trabajo de manera rápida y flexible.
- ¿que es un “branch” o “rama”?
  - Una **línea de desarrollo**, es decir, un conjunto de artefactos de una versión o actualización de un producto. → “la rama del req01”
- A diferencia de los tags, que simplemente son una referencia a un commit en particular, un branch es un **nuevo camino** de versiones en un historial separado que luego de varios commits se puede unir (merge) con otro branch.

Ram

- Por defecto la rama, “master” es la que se usa para los commits.
  - **Una buena práctica es que nunca un desarrollador desarrolle sobre el master.**
- Ante un nuevo commit el master avanza el puntero al siguiente Sha1.
- Así el master siempre apunta al último commit de dicha rama.
- Y cada commit siempre tiene un puntero al commit anterior lo cual puede ser de utilidad cuando realizamos un merge o reorganizamos el código.



Organizar el proyecto en ramas



Tarea 03

- En el repositorio local crear las rama "develpop" y "release"
  - `git branch develop`
  - `git branch release`
- Verificar que la rama ha sido creada, ejecutando el comando que lista las ramas.
  - `Git branch -l`
- Mostrar a que SHA1 apunta
  - `Git  show-ref`

Tarea 04

- En github crear un issue, llamado "REQ01"
- Luego en el repositorio local crear la rama "req01" basado en el branch "develop"
  - git branch req01 develop
- Verificar que la rama ha sido creada, ejecutando el comando que lista las ramas.
  - Git branch –l
- Mostrar a que SHA1 apunta
  - Git show-ref
- Traer a nuestro directorio de trabajo la estructura de la rama creada y construir una versión desde allí
  - git checkout req01

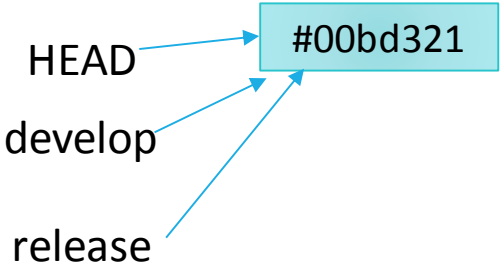
Tarea 05

- Agregar al final de archivo01.txt
  - linea 19 (req01)
- Agregar al index el archivo y realizar un commit con el siguiente mensaje "closes #1"
  - git add .
  - git commit –m "closes #1"
- Ver estado de los branches con el comando "git branch –v"
  - Muestra las ramas locales y a que SHA1 apunta cada una.
- Visualizar los cambios con develop
  - git diff develop

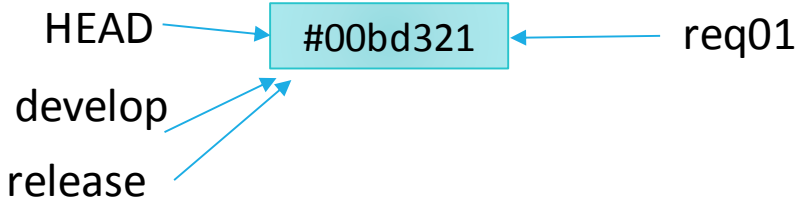
Analisis de evolución

```
$ git commit -m "Commit inicial"
```

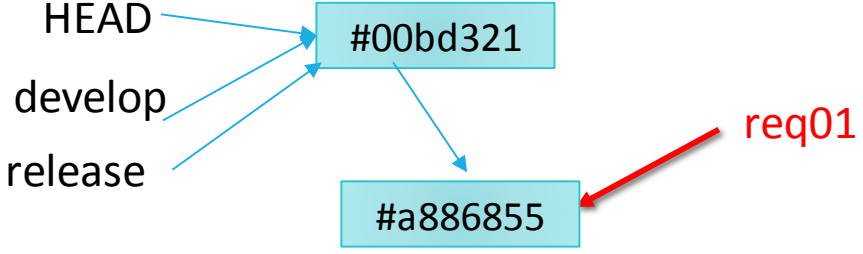
```
$ git branch develop
$ git branch release
```




```
$ git branch req01
$ git checkout req01
```



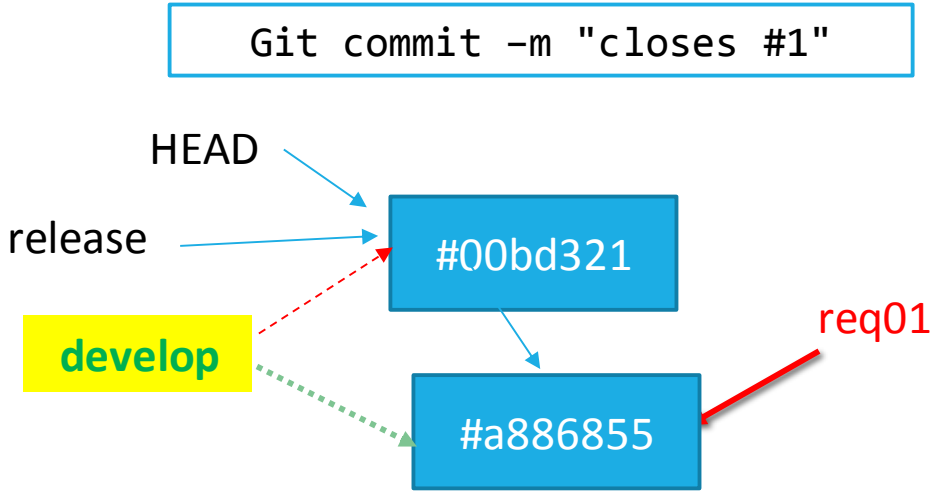

```
$ git commit -m "closes #1"
```



Tarea 06: Integrar los cambios

- Tenemos cambios realizados en la rama "req01" y queremos llevarlos a la rama "develop".
- Gráficamente es solo realizar lo siguiente

```
#ir a la rama develop
$ git checkout develop
#mezclar la rama develop con la rama req01
$ git merge req01
    Actualizando 00bd321..a886855
    Fast-forward
    archivo01.txt | 1 +
    1 file changed, 1 insertion(+)
$ git log --oneline –graph
#enviar los cambios al servidor
$ git push origin develop
```



Merge

- El comando "merge", mezcla el código de un Branch que pasamos como parámetro al branch actual.
- No se pueden realizar merges si tenemos cambios sin “commitear”.
- Luego de esto si el merge es exitoso, tenemos una copia local con datos modificados que deben ser agregados al staged y comiteados.
- Cuando no hay conflictos los archivos se mezclan automáticamente en el directorio de trabajo y se pasan al staging y al respositorio con la versión nueva.
- Cuando hay conflicto el usuario tiene que editar entonces las versiones en el directorio de trabajo, pasarlas manualmente al stage y comitear.

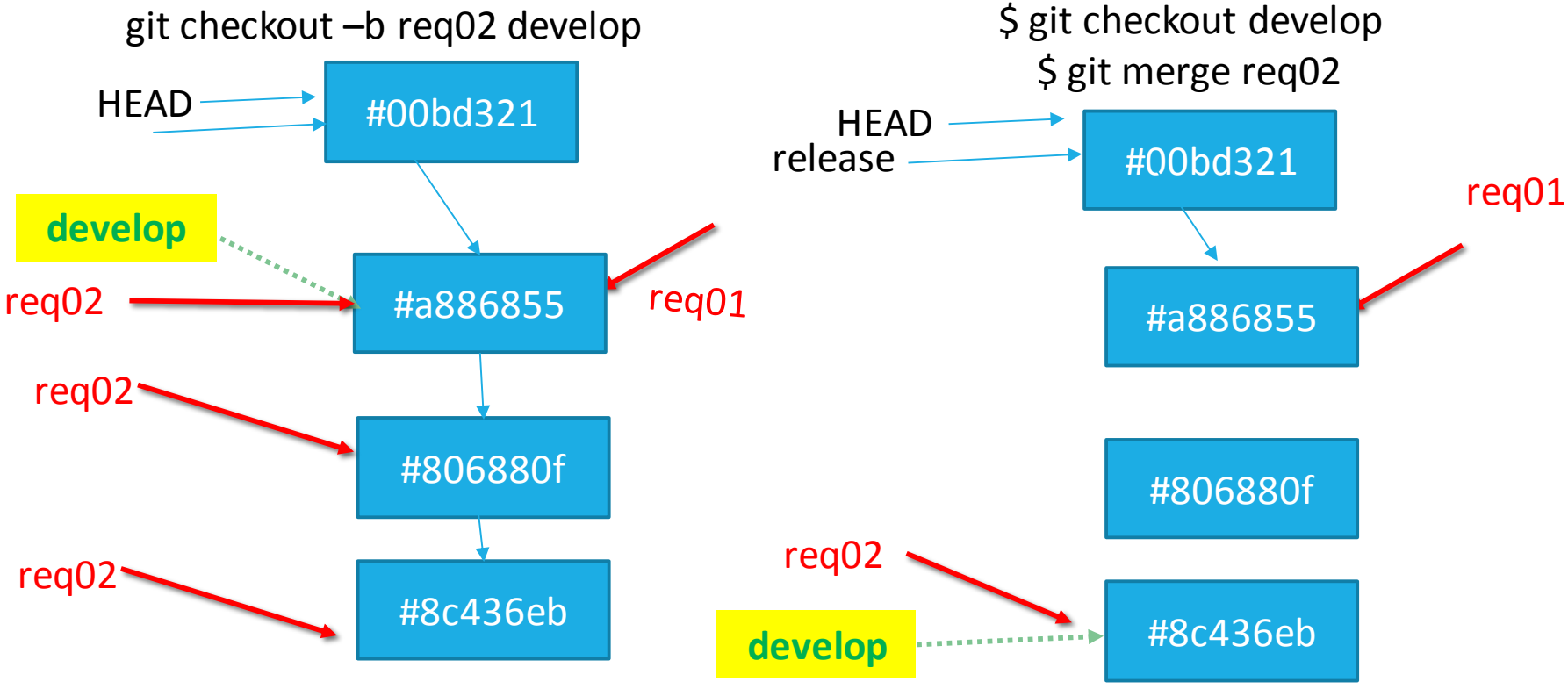
Merge

- La estrategia por defecto para el merge es “Fast-forward”
- Ocurre si el merge implica agregar cambios a partir de la versión de la rama desde la que estamos realizando el merge
  - **No hay cambios en el branch de destino que no estén en el branch de origen**
  - No obstante **el origen si puede tener cambios nuevos** a agregar.

Tarea 07: crear req02

- Ir a la gestión de issues de github y Crear el issue "req02"
- Verificar que el issue req01, sigue abierto pero con un mensaje que lo asocia al commit anterior (https://help.github.com/articles/closing-issues-using-keywords/)
  - El issue se cerrará cuando pasemos el commit a una rama master.
- Crear la rama "req02" con el comando abreviado
  - **$ git checkout –b req02 develop**
  - Esto crea la rama "req02" desde develop, y realiza el checkout a dicha rama.
- Modificar archivo01.txt (agregar una linea)
  - **agregar cambios al stage y realizar commit con el mensaje "arreglo #2"**
- Modificar archivo03.txt (agregar una linea)
  - **Agregar cambios al stage y realizar commit con el mensaje "closes #2"**
- Ver el historial de la rama actual con ”$ git log --oneline –graph”

Análisis de la situación



Tarea 08: integrar en release y en master.

- Ya hemos solucionado los primeros 2 requerimientos del primer release.
- Armar una nueva versión en la rama release, que será la que se use para hacer las pruebas finales.
  - **$ git checkout release**
  - **$ git merge release develop**
- Enviar las ramas al servidor remoto
  - **$ git push origin release**
  - **$ git push origin develop**
- Además etiquetar el estado actual de release, como "RC-v01" así sabemos que esta combinación de archivos corresponde al "Release candidate de la version 01".
  - **$ git tag "RC-v01"**
  - **$ git push –tags**

Tarea 08: integrar en release y en master.

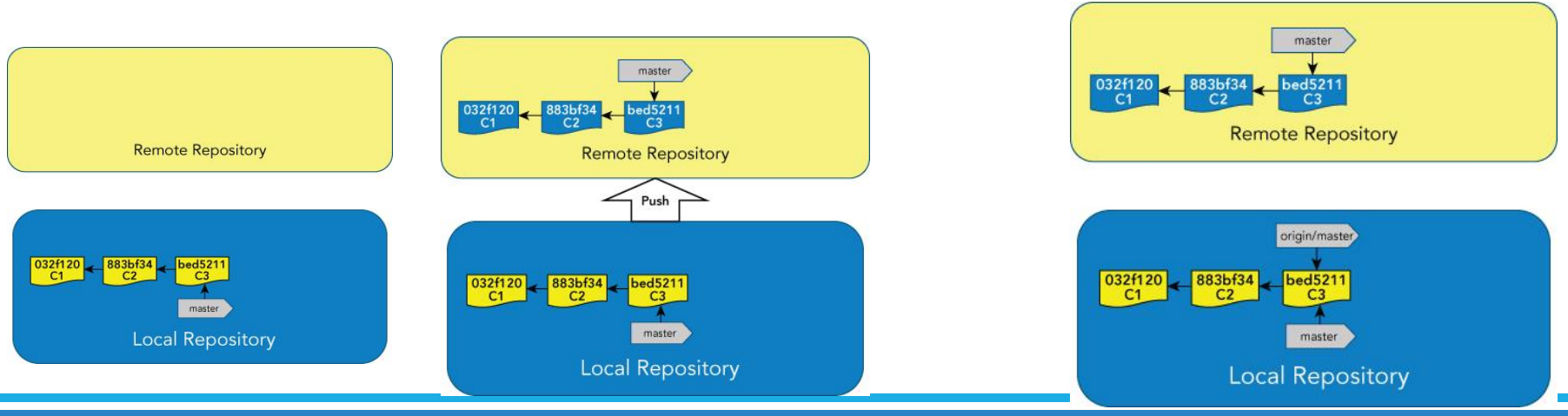
- Luego de haber probado el RC-v01, podemos pasarlo a producción, por lo tanto vamos a realizar un merge, entre "release" y "master" y luego realizar un tag con "V01" en master
  - **$ git checkout master**
  - **$ git merge master release**
  - **$ git push origin master**
  - **$ git tag v01**
  - **$ git push –tags**
- **Verificar que se cerraron los 2 requerimientos con los commits!!!**

Interacción con repositorios remotos

- Visualizar el estado de las ramas remotas con git branch -r (r: remoto)
  - Ejecutar git branch –r para visualizar ramas remotas
- Para mostrar ramas locales y remota se usa el modificador -a
  - $ git branch -a

Interacción con remotos: push

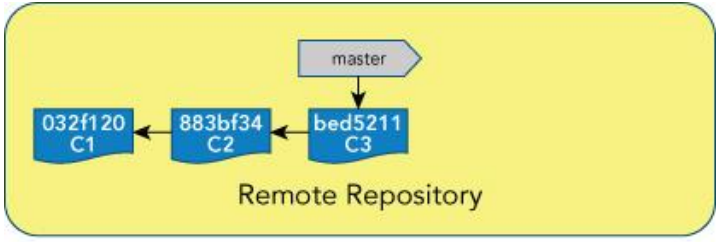
- El comando push permite enviar cambios locales al repositorio remoto en la rama que estamos definiendo
  - Ejemplo, comenzamos con un repositorio remoto vacío y clonado
  - Agregamos contenido al repositorio local y realizamos un push
  - Git crea un puntero “origin/master” para indicar donde apunta la rama actual en el repositorio remoto.



Revisar status: “local” vs “remoto”.

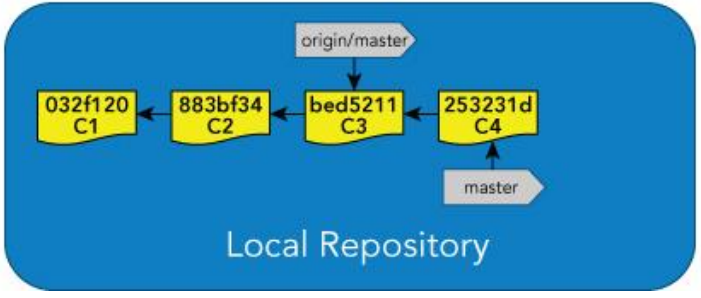
- Supongamos que realizamos un commit en el branch local. Podemos ejecutar el comando status y ahora nos indicará cual es el estado de la rama local en función de la remota.

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```



Tener en cuenta que esta diferencia solo se calcula en base a la última vez que se realizó la sincronización con el repositorio remoto.

Si se intenta volver a sincronizar puede ser que el repositorio remoto también haya avanzado.

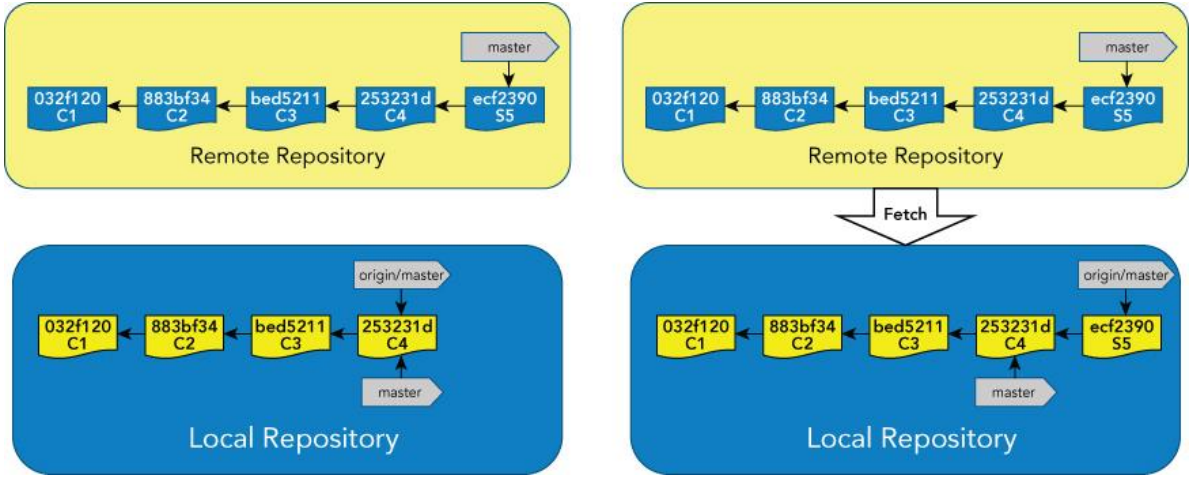


Formato del comando push

- El comando push espera como argumentos una referencia a un **repositorio remoto** y una referencia a la **rama remota** donde enviar los cambios de la rama de trabajo actual *(o si queremos pushear desde otra rama local, lo podemos indicar también como un tercer argumento pero es menos común).*
- El caso más común
  - $ git push <remote repository> <remote branch>
- Si queremos pushear un branch especifico
  - $ git push <remote repository> <local branch>:<remote branch>
- Por defecto, push rechaza todos los cambios que no pueden realizarse mediante un mezclado (merge) fast-forward.

fetch

- Una vez que un repositorio es clonado, se establece una conexión entre ramas remotas y locales y mediante el comando “fetch” se pueden obtener las últimas actualizaciones de las ramas remotas
- Formato: git fetch [<options>] [<repository> [<refspec>…]]



Fetch

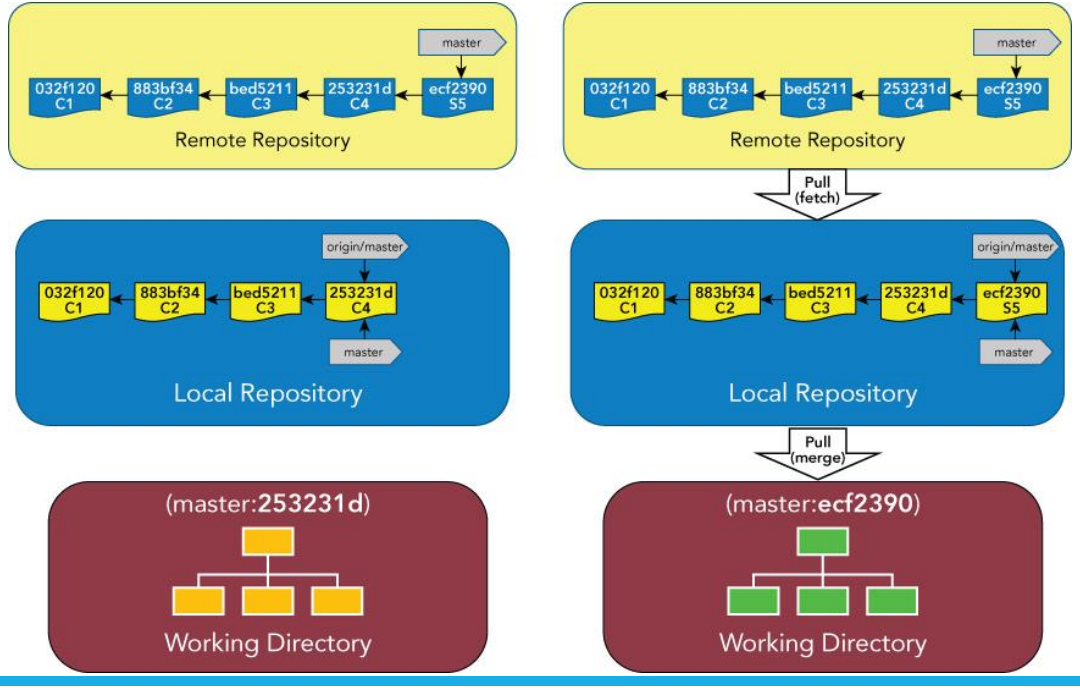
- Si analizamos la imágen, hemos subido al repositorio un cambio, pero luego alguien agrego al repositorio otro cambio y lo hemos descargado, **pero no en la rama de trabajo actual!!**
- Resultado de status **antes de realizar el fetch**
  - `$ git status`
  - `On branch master`
  - `Your branch is up-to-date with 'origin/master'.`
  - `nothing to commit, working directory clean`
- Porque dice que está actualizado con “origin/master” ? porque compara el hash del puntero “origin/master” con el último hash de origin master remoto descargado, **y coinciden!**

Fetch

- Resultado de status **luego del fetch**
  - `$ git status`
  - `On branch master`
  - `Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.`
  - `  (use "git pull" to update your local branch)`
  - `nothing to commit, working directory clean`
- Luego del fetch la comparación muestra obviamente que nuestra rama “master” está un commit por detrás de la rama “origin/master”.
- Tenemos que realizar el merge.

Pull

- El comando pull permite en una misma operación realizar un fetch y un merge, siempre que el **merge fast-forward** sea posible.



Tarea 09: borrar las ramas locales. Crear 2 issues

- Una buena práctica es borrar las ramas locales una vez que han sido integradas a la rama "develop"
  - **`$ git branch -d req01`**
  - **`$ git branch -d req02`**
- Crear el issue "req03"
  - Crear localmente una rama "req03" desde develop.
  - Modificar el archivo "archivo00.txt" agregando una linea
  - Agregarlo a staging y realizar un commit con el mensaje "req03 #3 parte 1"
- Crear el issue "urgente urgente explota producción "
  - ¿Qué hacemos? Tenemos que corregir un problema del código en producción y tenemos un requerimiento comenzado y a medio terminar!!!

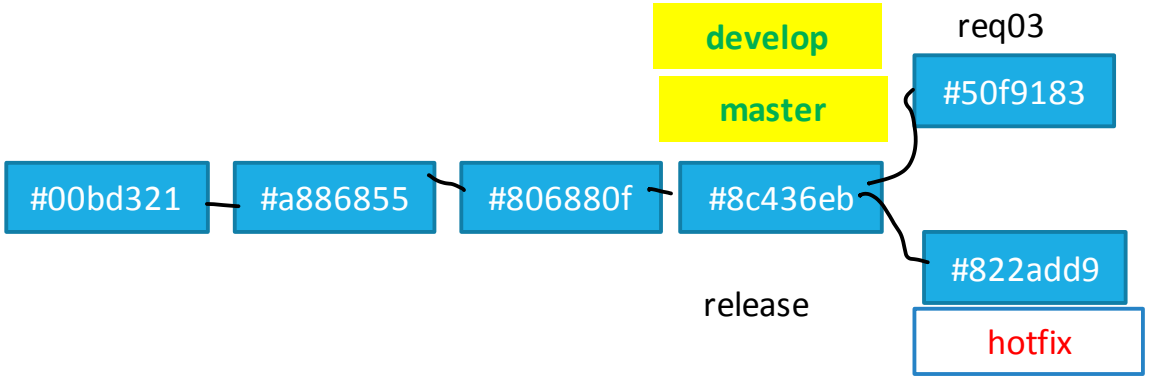
Tarea 10: resolver el hotfix

- Crear una rama "hotfix" desde la última versión del "master"
  - git checkout -b hotfix master
- Verificar las diferencias entre hotfix y req03
  - git diff hotfix req03
- El hotfix implica agregar al final de "archivo02.txt": "linea 29"
  - Agregar al staging y realizar un commit con el mensaje "fix #4"
  - git tag RC-v01-hf1
  - git push origin hotfix
  - git push --tags

Tarea 10: resolver el hotfix

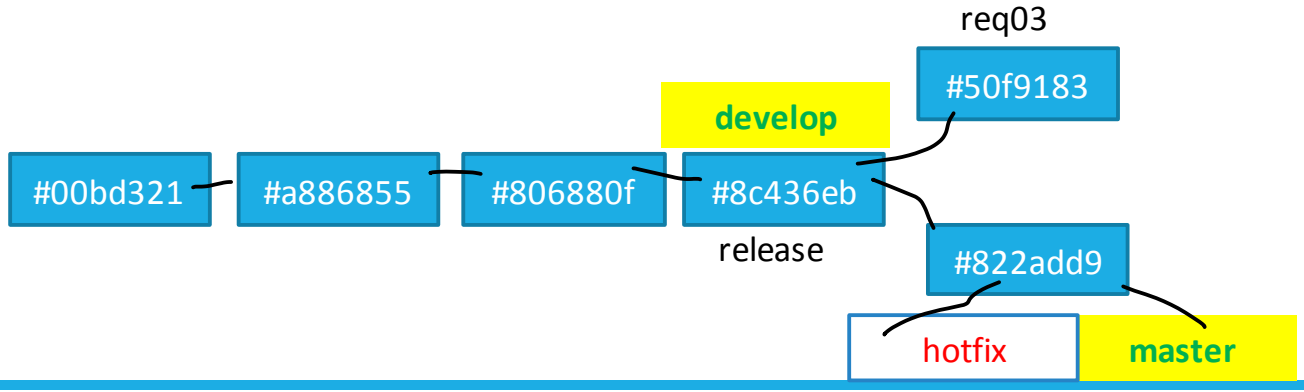
- Probamos el sistema con la rama hotfix, funciona correctamente, por lo tanto enviamos los cambios a la rama master
  - git checkout master
  - git merge master hotfix
  - git tag v01.1
  - git push origin master
  - git push --tags

Analisis de la situación



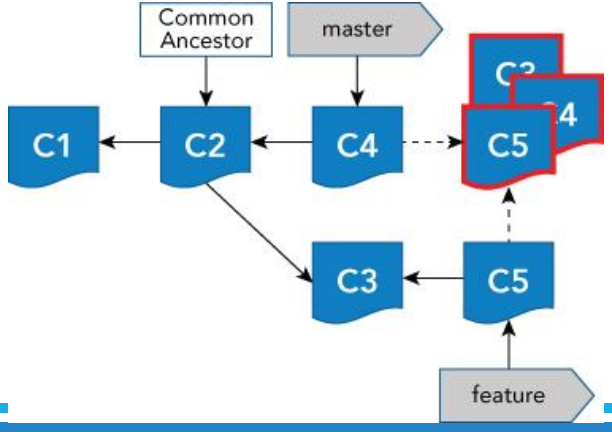
Como me llevo el hotfix al req03?
Tengo 2 ramas, 2 caminos!!!

```
$ git checkout master
$ git merge master hotfix
```

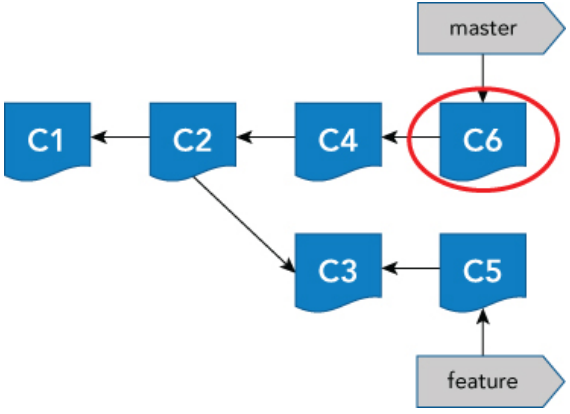


Three way merge

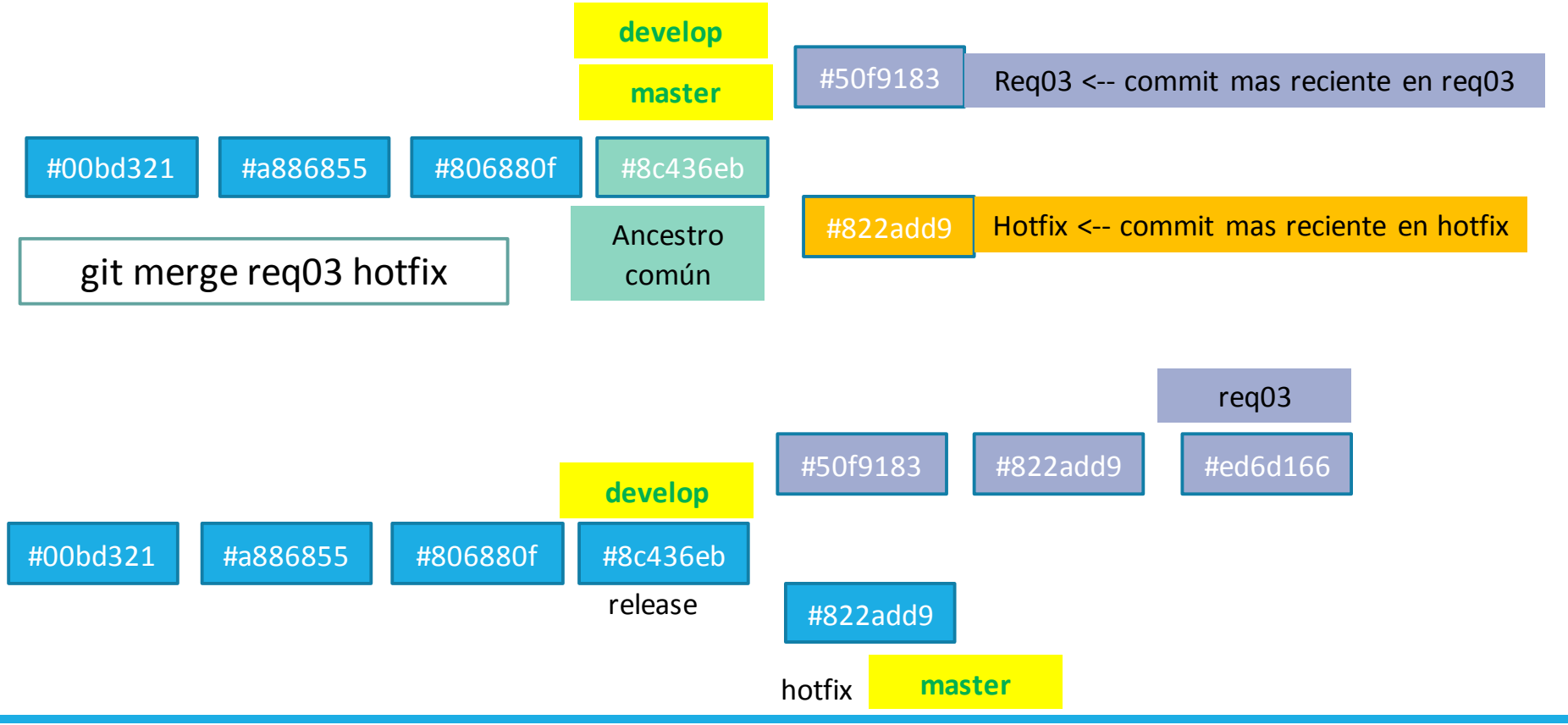
- Cuando se da está situación git busca 3 piezas de información
  - El comit más reciente del origen
  - El commit más reciente del destino
  - El más reciente ancestro en comun
- De estos 3 commits git interpola como realizar un merge y lo intenta
  - Si es exitoso crea un nuevo commit con el resultado llamado merge commit.
  - Si no es exitoso debemos realizar la mezcla de manera manual



El mezclado automático es existoso.



Analisis de la situación



Tarea 11: volver a req03

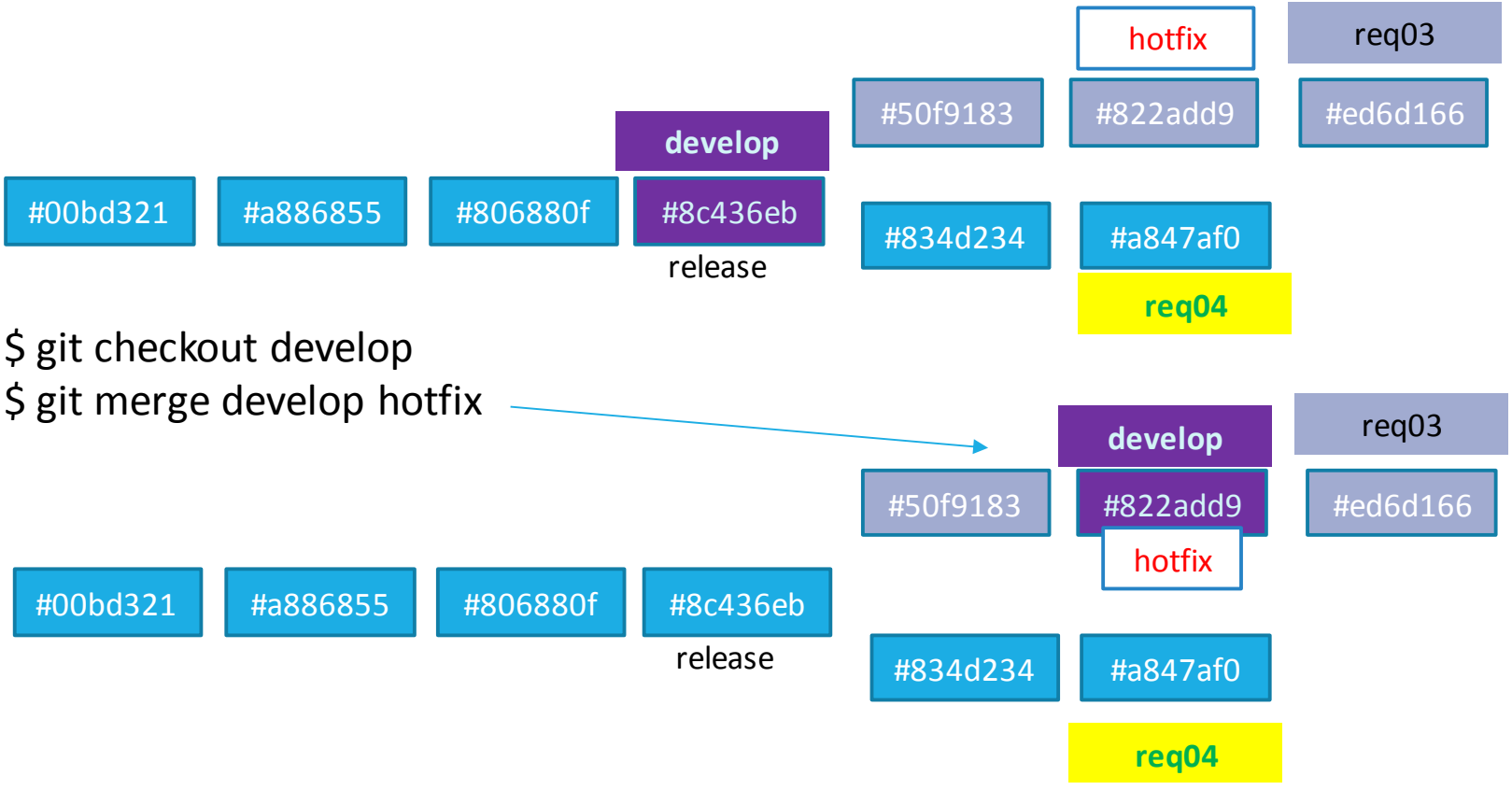
- Ahora volvemos a req03, **pero si no traemos la última corrección que hicimos en producción, generaremos una versión que tiene un bug importate** .
  - Git checkout req03
  - Git merge req03 hotfix
    - Merge made by the 'recursive' strategy.
    - archivo02.txt | 3 ++-
    - 1 file changed, 2 insertions(+), 1 deletion(-)
  - git log --oneline --graph

```
*   ed6d166 (HEAD -> req03) fusion Merge branch 'hotfix' into req03
|\
| * 822add9 (tag: v01.1, origin/master, origin/hotfix, master, hotfix) fix #4
* | 50f9183 req 03 #3 parte 1
|/
* 8c436eb (tag: v01, tag: RC-v01, origin/release, origin/develop, req02, release, develop) closes #2
* 806880f trabajo en #2
* a886855 (req01) closes #1
* 00bd321 commit inicial
```

Tarea 12

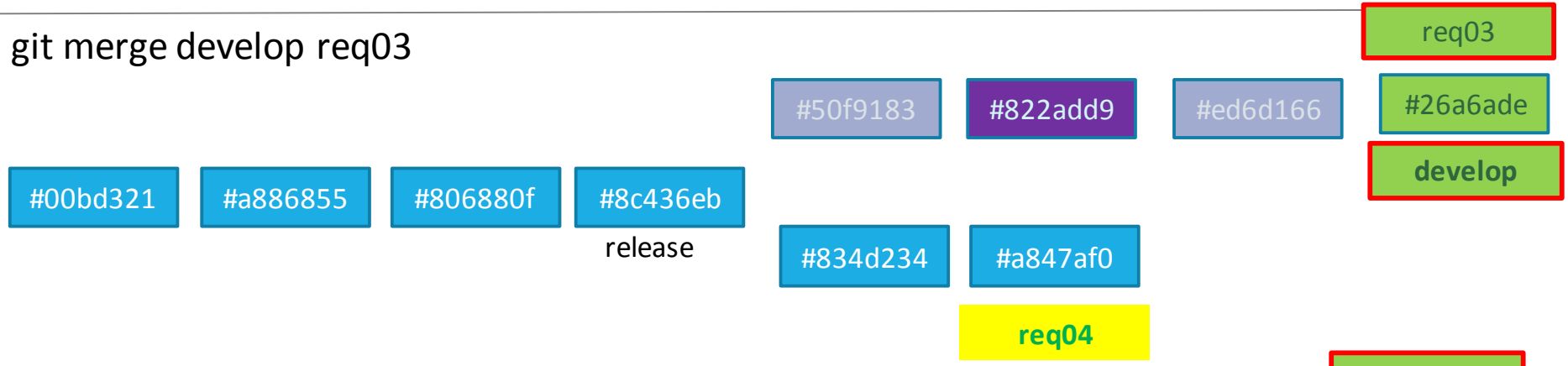
- Continuar trabajando en el req03. Modificar archivo01.txt agregar 1 linea y borrar la primer línea y agregar "linea 14 (req03)"
- Realizar un git add . y git commit –m "fix #3 "
- Revisar las diferencias con develop
- Crear un issue "req04" y crear la rama local "req04" y comenzar a trabajar en ella
  - Git checkout –b req04 develop
  - Abrir archivo01.txt y
  - Borrar "linea 18"
  - Agregar luego de linea 16 : "linea 17 req04"
  - Modificar linea 19 por "linea 19 req04"
  - Realizar git add . y git commit –m "solucion en #5 req4"
- Agregar el archivo archivo04.txt con 2 lineas de codigo "linea 40" y "linea 41"
  - Realizar git add . y git commit –m "solucion en #5 req4 parte 2"

Analisis de la situación - Merge en develop

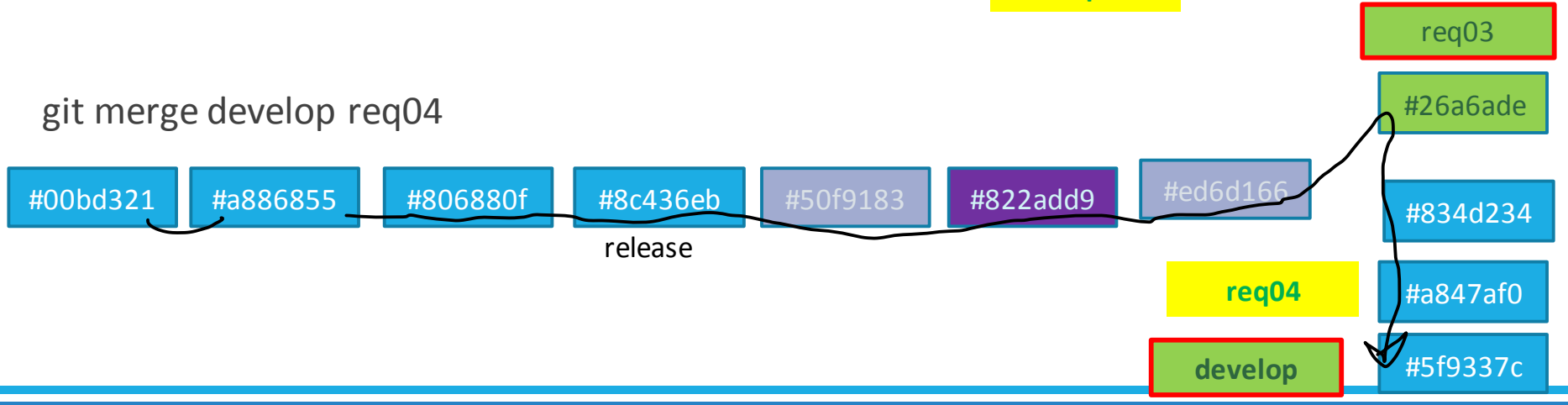


Analisis de la situación - Merge en develop

git merge develop req03



git merge develop req04

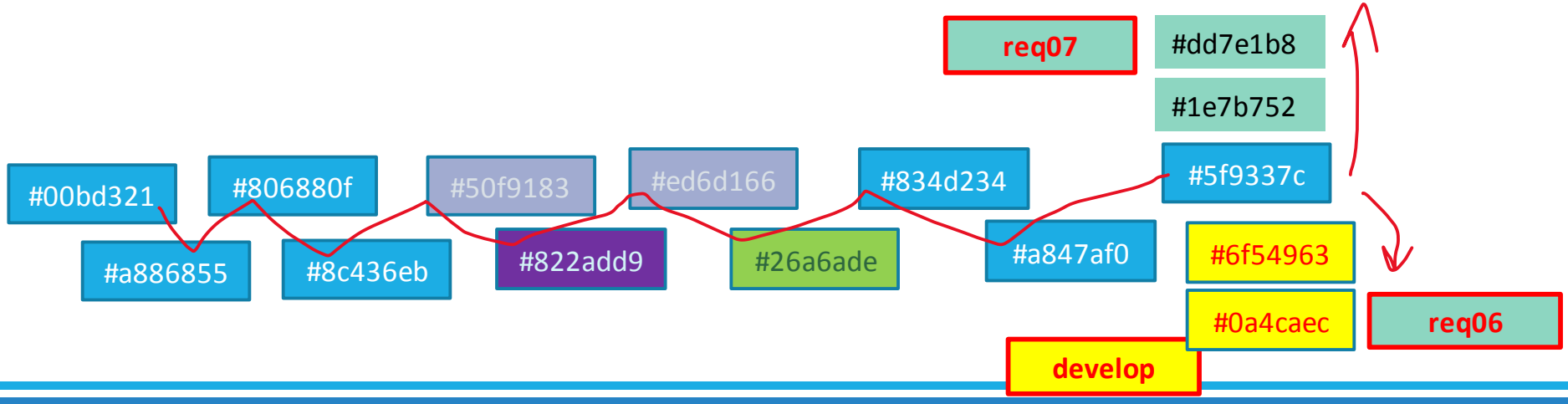


Tarea 13: desarrollar 2 ramas en simultaneo

- Integrar todos los cambios a anteriores a las ramas: release y Master
  - Generar los tags
  - Realizar el push al servidor
- Crear 2 requerimientos y avanzar en simultaneo
  - Crear el requerimiento "req 05" y el branch para resolverlo.
  - En dicho branch modificar archivo00.txt y a linea03 reemplazar por "linea 03 req05 abc"
  - Realizar un git add . Luego git commit -m "trabajo parcial en #6"
  - Crear el requerimiento "req 05" y el branch para resolverlo.
  - En dicho branch modificar archivo00.txt y a linea03 reemplazar por "linea 03 req06 xyz"
  - Realizar un git add . Luego git commit -m "trabajo parcial en #6"
  - Volver a la rama req05, modificar archivo04.txt agregar "linea 98" al final
  - Realizar un git add . Luego git commit -m "closes #6"
  - Volver a la rama req06, modificar archivo03.txt agregar "linea 99" al final
  - Realizar un git add . Luego git commit -m "closes #7"

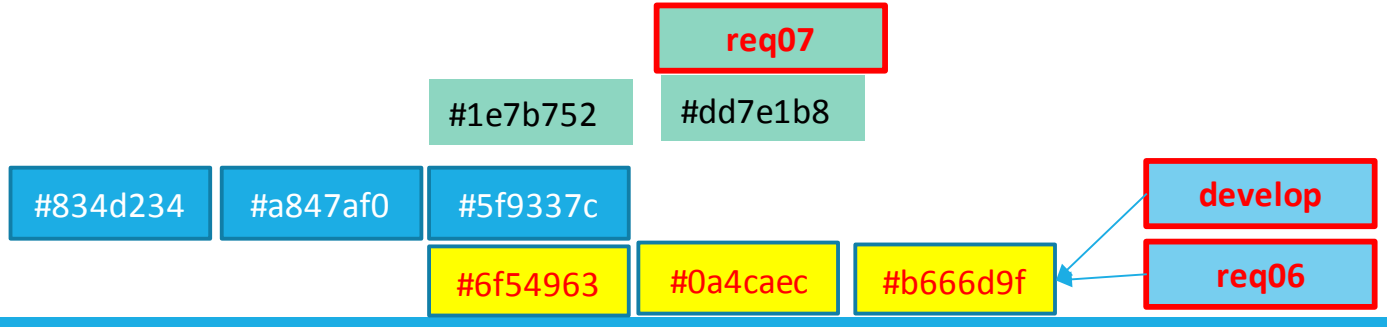
Tarea 14: integrar req05 en develop

- Ir a la rama develop
  - Git checkout develop
- Realizar el merge con la rama req05
  - Git merge develop req05



Tarea 15: reabrir req05 y corregir archivo00.txt

- Volver a req05 y reabrir archivo00.txt
- Modificar
  - Linea 03 req05 abc 999
  - Linea 04 req05 123
- Realizar "git add ." Luego "git commit –m 'fix #6".
- Ir a la rama develop y realizar un merge



Tarea 16: realizar un merge y obtener conflicto

- En este caso el mezclado no puede ser automático porque archivo00.txt tiene diferentes versiones en las 2 ramas y la historia no se puede diferenciar.
  - Primer fue modificado en la rama5, luego en la 6, luego en la rama 5.
- **`git merge develop req06`**
  - `Auto-fusionando archivo00.txt`
  - `CONFLICTO (contenido): Conflicto de fusión en archivo00.txt`
  - `Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.`

Conflictos

- Un primer aspecto a tener en cuenta: “MERGE es un ESTADO”.
  - En este estado, git deshabilita los cambios de contexto hasta que la operación es completada.
  - Si todo mergea automáticamente, no notamos ningún cambio de estado
  - Si hay conflictos se requiere intervención del usuario: no se puede realizar otra operación *(ej: cambiar de rama)*, hasta resolver conflictos o abortar la operación
- Si no deseamos solucionar los conflictos una opción es abortar
  - $ git merge --abort

Tarea 17: resolver el conflicto

- Ejecutar "git status"

```
En la rama develop
Tienes rutas no fusionadas.
  (arregla los conflictos y corre "git commit"
  (usa "git merge --abort" para abortar la fusion)

Cambios a ser confirmados:

        modificado:     archivo03.txt

Rutas no fusionadas:
  (usa "git add <archivo>..." para marcar una resolución)

        ambos modificados:      archivo00.txt
```

Tarea 17: ver el contenido de archivo01.txt

- linea 01
- linea 02
- <<<<<<< HEAD
- linea 03 req05 abc 999
- linea 04 req05 123
- =======
- **linea 03 req06 xyz**
- **linea 04**
- >>>>>>> req06
- linea 09

- Las marcas que agrega git a los archivos donde detecta conflictos son ≪ y ≫
- La idea es que el usuario resuelva los conflictos dejando el archivo en una versión final apropiada y luego pase a staging este nuevo archivo junto con los archivos que ya estaban en staging que no tenían conflicto.
- ¿Podemos hacer un commit de los archivos que no tienen conflicto, que ya están en staging, y después resolver los que tienen conflicto? **NO, es una operación atómica.**
- **Git está en estado "merging" y por lo tanto hasta no salir de dicho estado no admite commits en el repositorio.**

Tarea 17: ver el contenido de archivo01.txt

- linea 01
- linea 02
- **linea 03 req05 abc 999 req06 xyz**
- **linea 04 req05 123**
- linea 09

Ahora que generamos una nueva version de archivo00.txt que integra las dos versiones, lo agregamos al staging y luego realizamos un commit

$ git add archivo00.txt

$ git status

$ git commit –m "merge dev y req 06"

```
mdominguez@mdominguez:~/Documentos/devs/caps/tallergit-02$ git add archivo00.txt
mdominguez@mdominguez:~/Documentos/devs/caps/tallergit-02$ git status
En la rama develop
Todos los conflictos resueltos pero sigues fusionando.
  (usa "git commit" para concluir la fusión)

Cambios a ser confirmados:

        modificado:     archivo00.txt
        modificado:     archivo03.txt
```

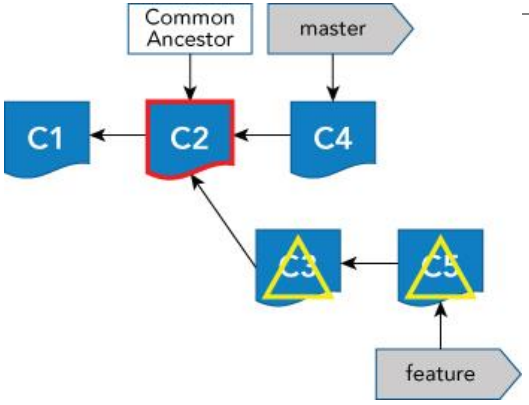
Tarea 17: resolver el conflicto

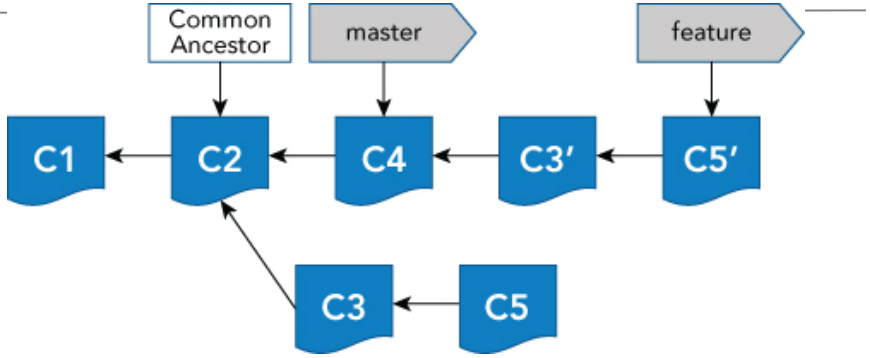
```
mdominguez@mdominguez:~/Documentos/devs/caps/tallergit-02$ git commit -m "merge req05 y req06"
[develop 026dbf5] merge req05 y req06
mdominguez@mdominguez:~/Documentos/devs/caps/tallergit-02$ git log --oneline --graph
*   026dbf5 (HEAD -> develop) merge req05 y req06
|\
| * dd7e1b8 (req06) closes #7
| * 1e7b752 trabajo parcial en #7
* | b666d9f (req05) fix #6
* | 0a4caec closes #6
* | 6f54963 trabajo parcial en #6
|/
*   5f9337c (tag: v02, tag: rc-v02, origin/release, origin/master, origin/develop, release, mas
|\
| * a847af0 (req04) solucion #5 req 4 parte 2
| * 834d234 solucion #5 req 4
* | 26a6ade (req03) correcion en #3 abc
* |   ed6d166 fusion Merge branch 'hotfix' into req03
|\ \
| * | 822add9 (tag: v01.1, origin/hotfix, hotfix) fix #4
| |/
* | 50f9183 req 03 #3 parte 1
|/
* 8c436eb (tag: v01, tag: RC-v01, req02) closes #2
* 806880f trabajo en #2
* a886855 (req01) closes #1
* 00bd321 commit inicial
```

Rebase

- Rebase es una alternativa al merge para integrar dos ramas.
- En general el efecto es el mismo, pero el proceso que se sigue, y el historial del árbol de commits es diferentes.
  - El merge de 3 caminos, busca los últimos dos commits de cada rama a unir y el último commit común y trata de unirlos.
- En un rebase, se identifica el último ancestro común y por cada cambio que posterior procede de la siguiente forma:
  - Se computa el “delta” del cambio, es decir que cambio con dicho commit. Para cada cambio, se intenta reaplicar el delta en el branch destino donde se está “mergeando”. **Intenta realizar los mismos cambios que se hicieron en la rama origen, desde el último commit la rama destino**
  - Si funciona sin conflictos, entonces la historia del branch se vuelve parte de la historia de la rama destino, desde el último commit.

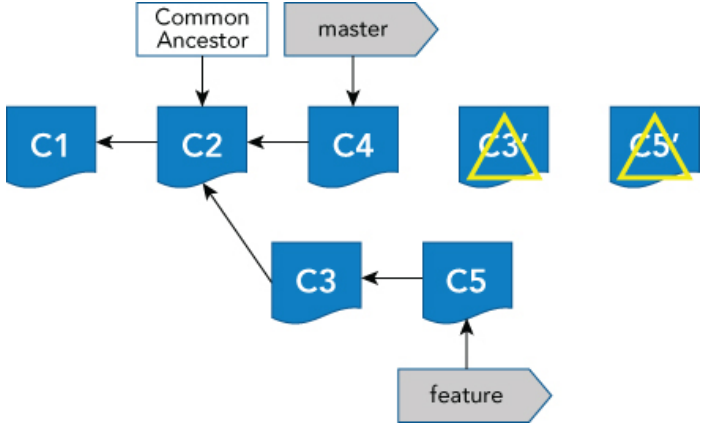
Rebase





Se calculan las diferencias que se hicieron en el feature desde el ancestro en común

Se arman los commits que representan el incremento.



Si se pueden aplicar sin conflict o ...se avanza la rama feature a "C5'"

C3 y C5 quedan huerfanos.

Tarea 18: generar un escenario de rebase

- Crear un nuevo branch desde develop "req20"
  - Modificar archivo01.txt, modificar "linea 12 (req20)" y "linea 15 (req20)"
  - Ejecutar "$git add ." Y "git commit -m "fix #20" "
- Crear un nuevo branch desde develop "req30"
  - $ git checkout develop
  - $ git checkout -b req30 develop
- Modificar archivo02.txt, modificar "linea 28 (req30)", y "linea 29 (req30)"
  - git add .
  - git commit -m "req30 parte 1"
- Modificar "archivo03.txt" cambiando "linea 99999"
  - git add .
  - git commit -m "req30 parte 2"
- git checkout develop
- git merge develop req30

Tarea 18: análisis de situación y realizar rebase

Quiero traer los últimos cambios de develop a "req20" para no generar conflictos. Pero el merge me va a generar una rama separada, porque ambas ramas ya avanzaron respecto del ancestro común

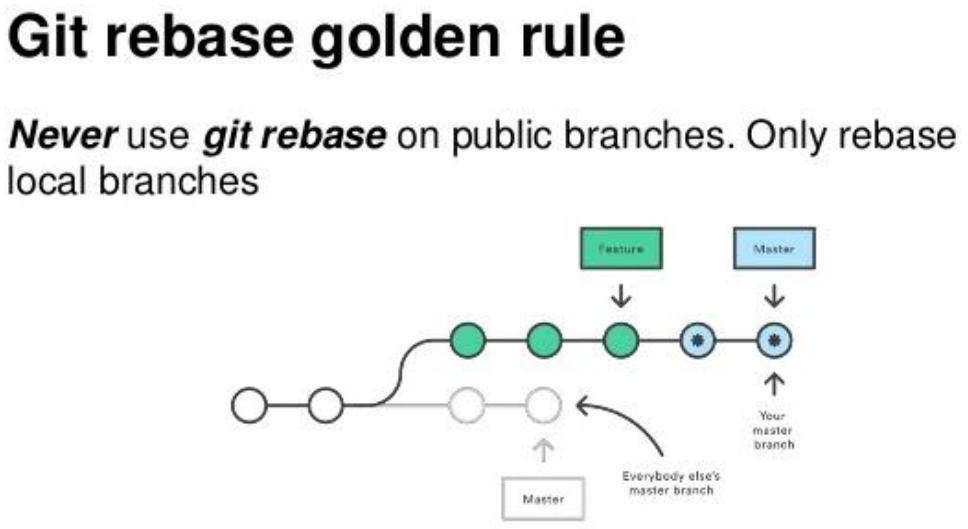
$git checkout req20
$ git rebase develop
$git log --oneline --graph

Si realizo un rebase, en #req20, reorganiza la rama, para aplicar los cambios de develo req20 en el orden que se hicieron desde el ancestro comun

Aplico los cambios de #1dca37b al último commit de develop y generó un nuevo comit

NO APLICAR REBASE A RAMAS PUBLICAS

- Las ramas de feature, como podemos apreciar, se crean y se destruyen en el directorio local.
- Las ramas que están en el servidor son: master, hotfix, develop, release.
- Si realizamos el rebase sobre cualquiera de estas ramas, estaremos cambiando la historia de dicho repositorio, de forma tal que el resto de los colaboradores no podrán reproducirla



Tarea 19: integrar en develop, en release y en master

- git checkout develop
- git merge develop req20
- git log --oneline --graph
- git checkout release
- git merge release develop
- git tag RC-v03
- git checkout master
- git merge master develop
- git tag v03
- git push origin master
- git push origin develop
- git push origin release
- git push --tags

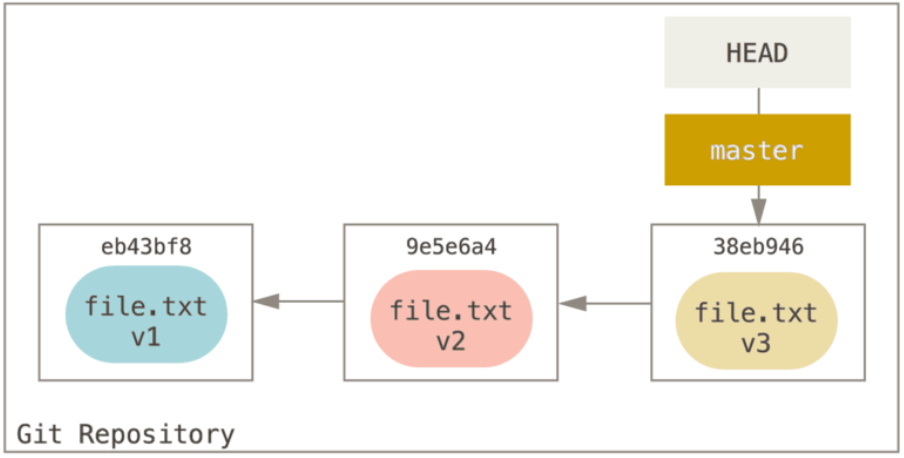
Resetear cambios

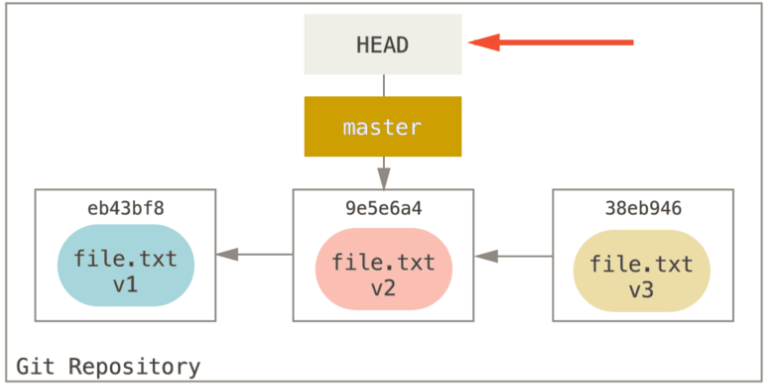
- El propósito de reset es realizar un rollback, y deshacer un conjunto de cambios, y volver al estado previo.
- Mueve el head del repositorio local a un estado previo y opcionalmente puede actualizar el staging y el directorio de trabajo con el contenido de dicho commit anterior.
- Hay 2 formas básicas de dicho comando
  - git reset [-q] [<tree-ish>] [--] <paths>…
  - git reset (--patch | -p) [<tree-ish>] [--] [<paths>…]
- Lo que hacen es actualizar el staging con la última versión del archivo, por lo que sale de staging dado que no tiene diferencias y vuelve a estado modificado.
- Es útil si agregamos al stage un archivo que no queremos comitear y lo queremos sacar de stage.

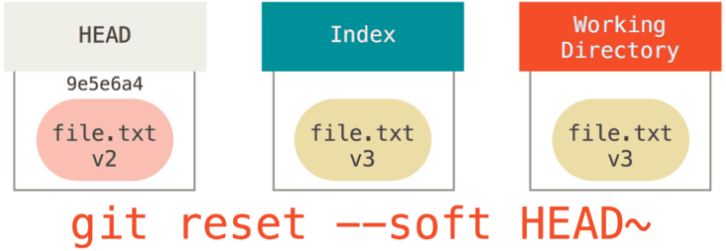
Resetear commits

- Una tercer variante de reset tiene 3 opciones: git reset [--soft | --mixed [-N] | --hard | --merge | --keep] [-q] [<commit>]
- Aquí se indica que parte del entorno local debe ser actualizada con el contenido del "commit" pasado como parámetro (SHA1 o TAG)
- Las opciones son 3
  - soft→ solo actualiza el HEAD del repositorio local con la nueva versión
  - mixed—actualiza el head y el área de staging (comportamiento por defecto).
  - Hard—actualiza HEAD, staging, y directorio de trabajo. Puee resetear completamente entorno local y volver al anterior commit en el respositorio local.
- Si queremos ir hacía atrás sin conocer los SHA1 podemos usar una sintáxis que resetea los punteros de manera relativ
  - $git reset --hard HEAD^ → 1 anterior
  - $git reset --hard HEAD^^^ → 3 anteriores
  - $git reset --hard HEAD~ → 1 anterior
  - $git reset --hard HEAD~3 → 3 anteriores

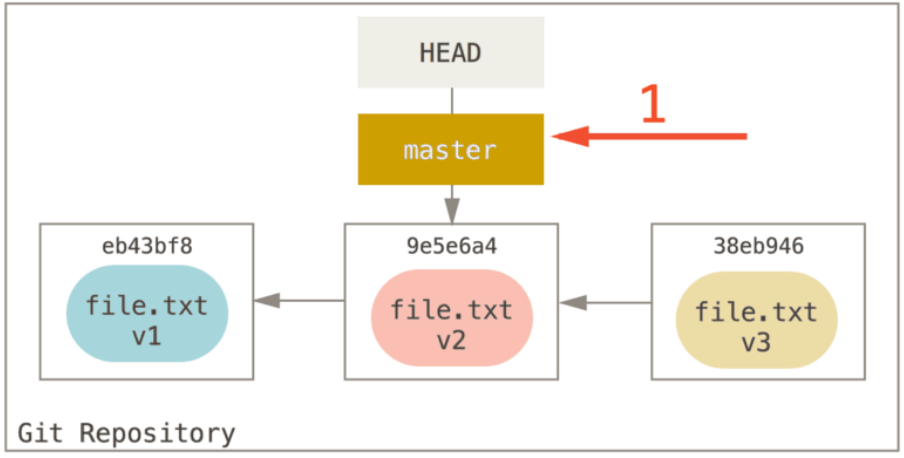
Resetear commits

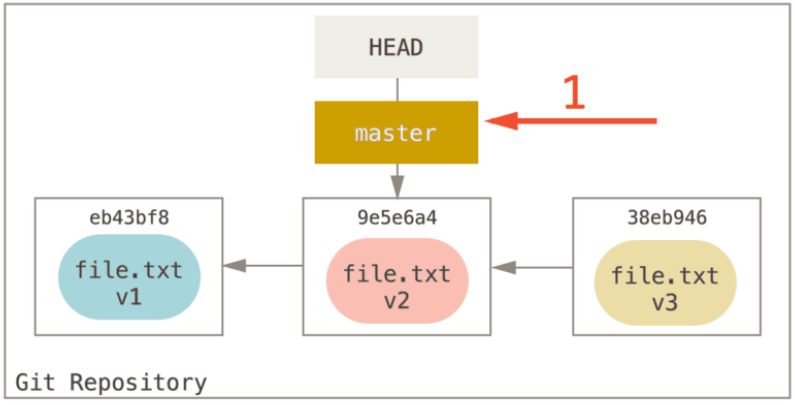


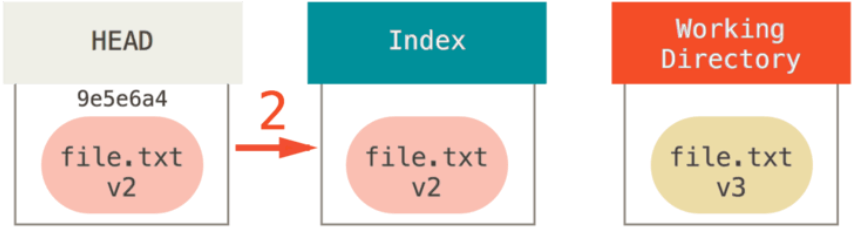




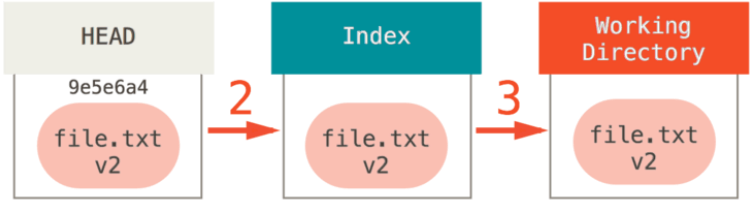
Resetear commits







git reset [--mixed] HEAD~



git reset --hard HEAD~

Tarea 20: Reset hard

- $ git checkout develop
- Editar archivo01.txt y modificar "linea 16 (test reset HARD)"
  - $ git add .
  - $ git commit –m "agrego cambio"
  - $ git log --oneline
- Deshacer lo que tenemos en staging con la última versión del HEAD(dejar la copia del directorio local solamente)
  - $ git reset --hard HEAD~
  - $ git log –oneline
    - Verificar que ahora fuims hacia atrás un commit.

Revert

- **Se recomienda no usar "reset" u operaciones que cambien la historia de ramas publicas. ¿Pero entonces como deshacemos cambios?**
- Usar git revert que trata de cancelar el efecto de los cambios realizados en un commit que deseamos descartar.
- Ejemplo
  - git revert HEAD^ → examina los commits y crea un nuevo commit que cancela nuestros cambios (o sea se crea una nueva version con 2 lineas borradas).
  - El revert agrega una version al final de la pila de versiones lo cual no causa el mismo problema que el reset, porque ahora el usuario que modifico el archivo tiene que descargar la nueva version, mergear localmente y subir su nueva version.

Tarea 21: revertir un cambio

- En la rama develop, abrir "archivo00.txt" y modificar una linea agregando al final "REVERTIR".
- Luego ejecutar
  - git add .
  - git commit -m "modifico"
  - git log --oneline
  - git revert HEAD~1
  - git log --oneline