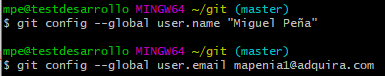
**Configurando Git**

Git busca el archivo. gitconfig en el directorio $HOME

Establecer tu nombre de usuario y dirección de correo electrónico

$ git config --global user.name "Miguel Peña"

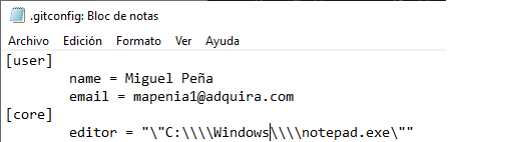
$ git config --global user.email [mapenia1@adquira.com](mailto:mapenia1@adquira.com)



Establecemos el editor, en nuestro caso el notePad

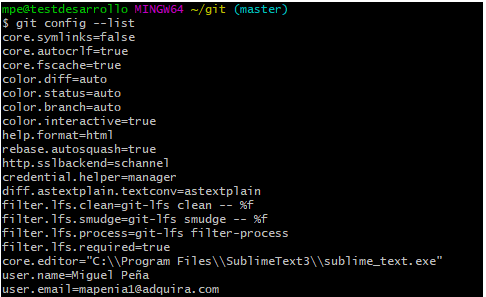
$ git config --global core.editor "\"C:\\\\Windows\\\\notepad.exe\""



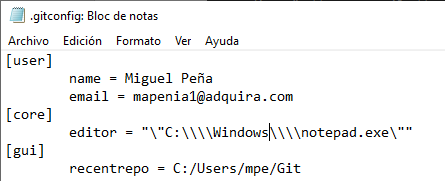


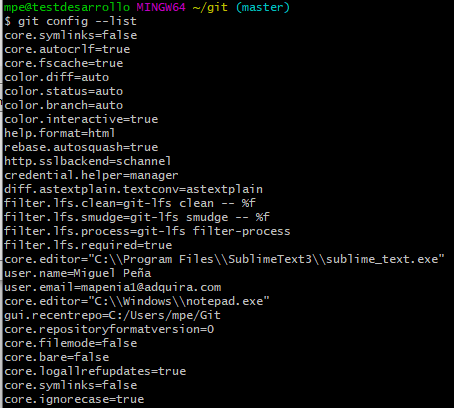
Comprobamos la configuración

$ git config –-list



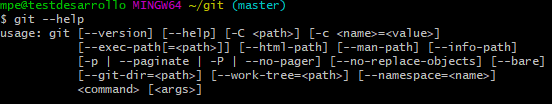
Como curiosidad decir que podemos editar el fichero ‘.gitconfig’ de nuestra configuración y modificarlo como texto plano y comprobamos que se modifican los datos sin problemas





## ¿Cómo obtener ayuda?

$ git –help



Ejemplo

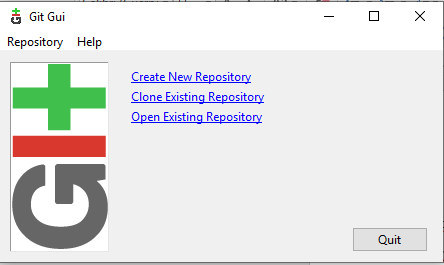


**Repositorios**

$cd carpeta/

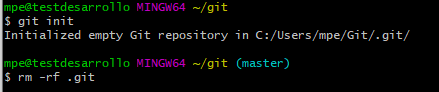
$git init -->Creamos un repositorio

También se pueden crear repositorios desde la herramienta grafica Git GUI

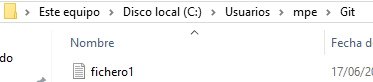


$cd carpeta/

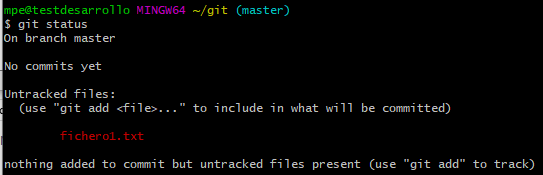
$rm -rf .git --> Eliminar un repositorio



Creamos un fichero con el bloc de notas mismamente y lo guardamos donde esta nuestro repositorio, en este caso C:\Users\mpe\Git

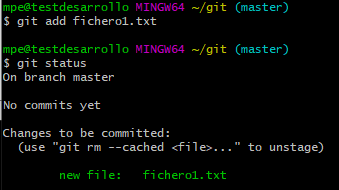


Comprobamos su estado



Vemos que ese fichero aún no está manejado por GIT, por lo que tenemos que añadirlo a nuestra área de almacenamiento (Staging area)

$ git add fichero1.txt



Ahora podemos ver que el fichero1.txt esta manejado por GIT y añadido al repositorio

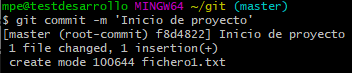
Para borrar este mismo fichero del repositorio lo haríamos mediante el comando

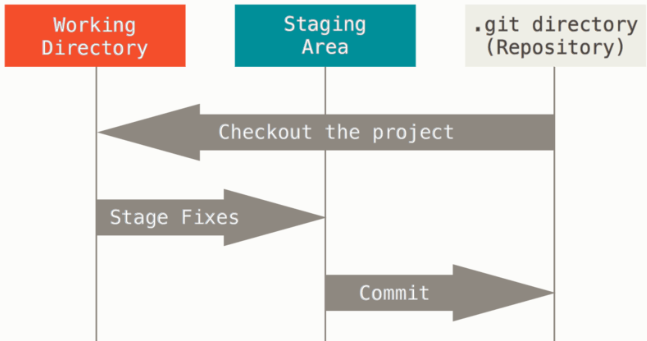
$ git rm --cached fichero1.txt



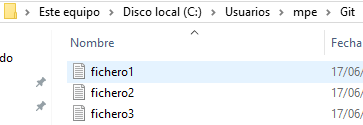
Una vez tenemos algo en nuestra área de almacenamiento vamos a proceder a realizar un commit y así confirmamos los cambios y los almacenamos en nuestro directorio GIT

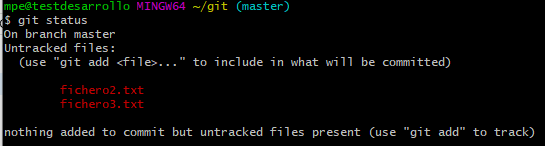
$ git commit -m 'Inicio de proyecto'

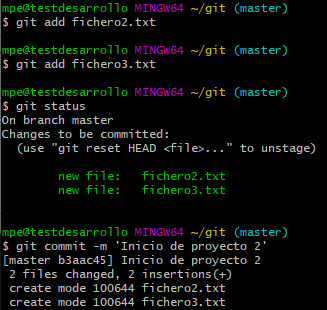




Ahora por ejemplo vamos a crear otros dos archivos y agregarlos al repositorio y hacemos un commit







Desde la herramienta Git GUI, podemos ver el historio de las versiones del repositorio

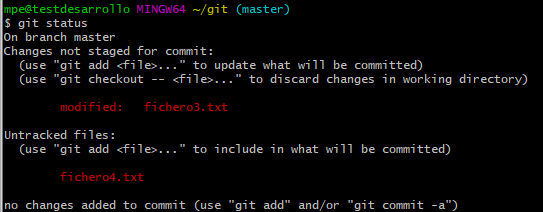


Si a continuación una vez hecho el commit comprobamos el estado del repositorio



Esto significa que tienes un directorio de trabajo limpio - en otras palabras, que no hay archivos rastreados y modificados. Además, Git no encuentra ningún archivo sin rastrear, de lo contrario aparecerían listados aquí

Ahora vamos a modificar un archivo, por ejemplo, el fichero3.txt y creamos el archivo fichero4.txt, al comprobar el estado, vemos que no hay ningún fichero preparado para confirmar

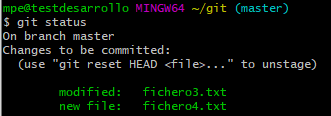


Vamos a sumar los dos ficheros al repositorio.

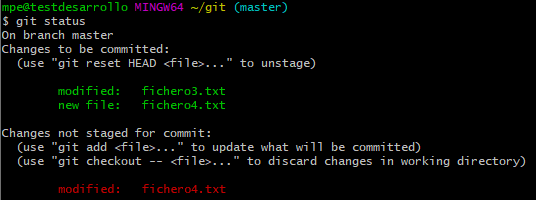
$ git add fichero3.txt

$ git add fichero4.txt

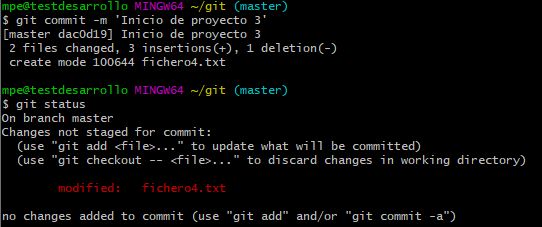
Y comprobamos que ambos ficheros están preparados para confirmar y almacenarse una copia en el directorio GIT



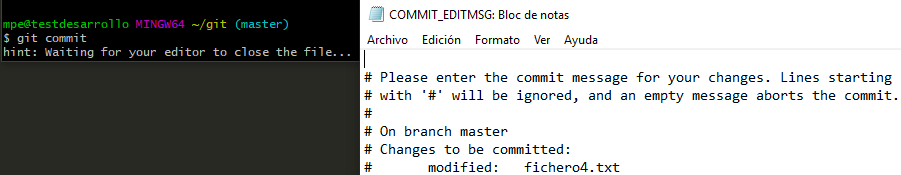
Antes de hacer el commit, para confirmar los cambios, vamos a modificar de nuevo uno de estos ficheros y comprobamos el estado

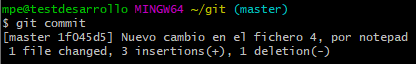


Si ahora hacemos un commit, el último cambio realizado no se subirá a nuestro repositorio GIT, ya que el fichero4.txt tiene la última versión de cuando se realizó el git add fichero4.txt

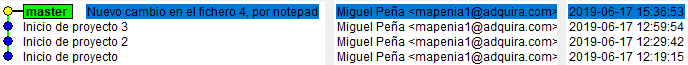


Si tenemos cambios pendientes para hacer el commit, podemos hacer un git commit y se abrirá el editor asociado en la variable core.editor



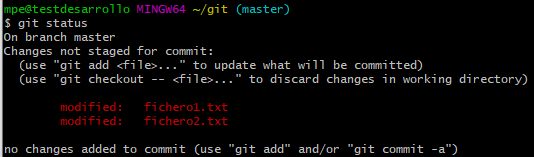


Vemos el mensaje en la herramienta Git GUI

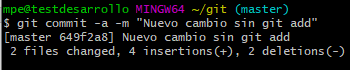


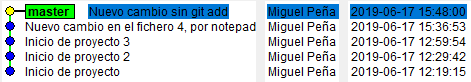
Existe la opción de saltarse el área de preparación, cuando por ejemplo modificamos un archivo y no queremos ejecutar la opción git add para dejarlo pendiente de hacer el commit.

Por ejemplo, modificamos dos ficheros.



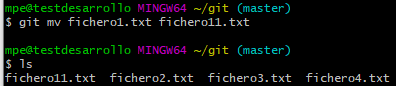
Ahora si hacemos un commit, ninguno de los dos ficheros se subiría a nuestro repositorio GIT, para eso tenemos la opción commit –a





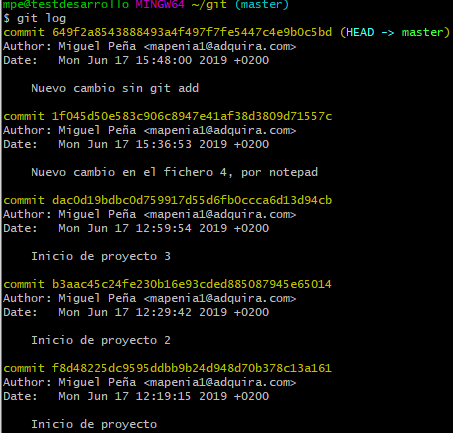
### Cambiar el Nombre de los Archivos

Para cambiar el nombre de un archivo existe el comando mv, git mv file\_from file\_to

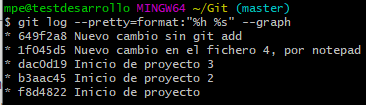


## Ver el Historial de Confirmaciones

Con el comando git log podemos ver el histórico de confirmaciones sobre un repositorio en orden cronológico, de más reciente a más antiguo



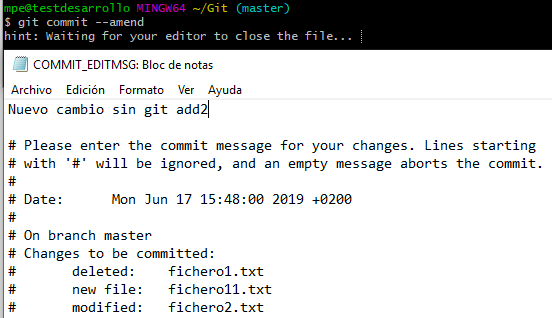
El comando git log tiene múltiples opciones para mostrarte exactamente lo que buscas

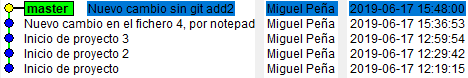


## Deshacer Cosas

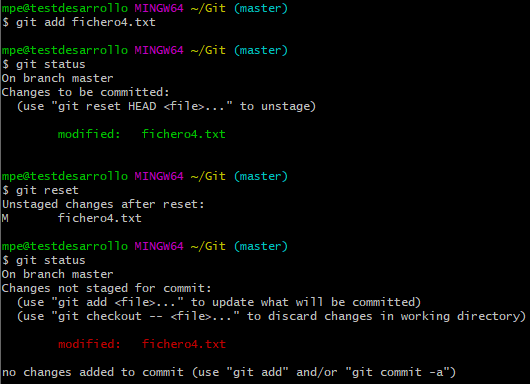
Aquí repasaremos algunas herramientas básicas usadas para deshacer cambios que hayas hecho.

Si quieres rehacer la confirmación, puedes reconfirmar con la opción --amend:





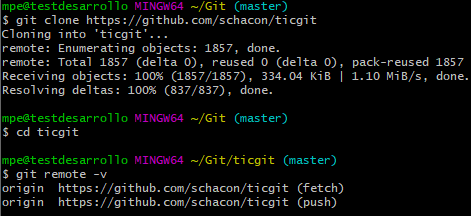
También podemos deshacer un cambio de estado de pendiente de actualizar a nos preparados para actualizar



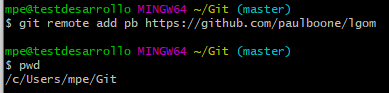
## Trabajar con Remotos

Para poder colaborar en cualquier proyecto Git, necesitas saber cómo gestionar repositorios remotos. Los repositorios remotos son versiones de tu proyecto que están hospedadas en Internet en cualquier otra red. Gestionar repositorios remotos incluye saber cómo añadir un repositorio remoto, eliminar los remotos que ya no son válidos, gestionar varias ramas remotas y definir si deben rastrearse o no, y más

Para ver mis servidores remotos, lo primero que vamos hacer es clonar mi propio servidor

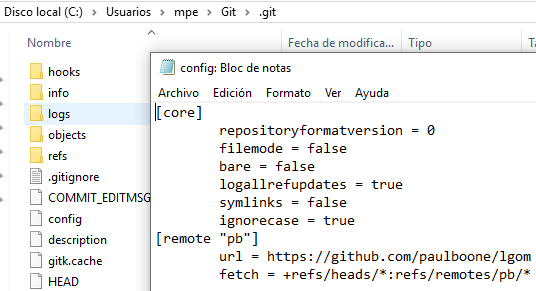


Para añadir remotos podemos hacer el siguiente comando





Vemos que la configuración de los remotos queda grabada en el fichero config



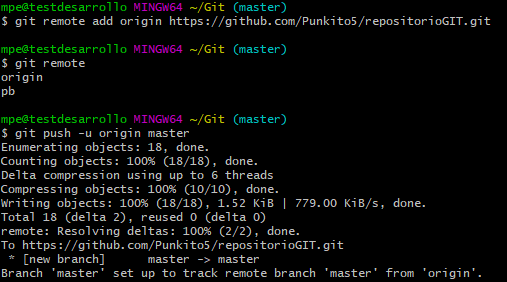
Como hemos visto, el remoto <https://github.com/paulboone>, está creado en la web github.com. Para ver cómo crear un servidor remoto en github.com, nos creamos una cuenta en dicha web y seguimos unos pequeños pasos.

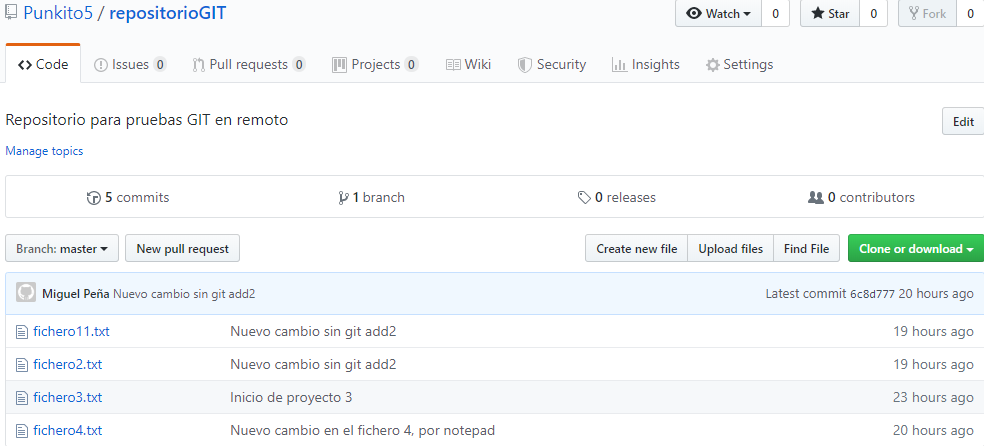
<https://desarrolloweb.com/articulos/crear-repositorio-git-codigo.html>

## Una vez creado el repositorio, subimos nuestro proyecto a Github con Push

$ git remote add origin https://github.com/Punkito5/repositorioGIT.git

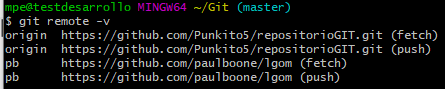
$ git push -u origin master





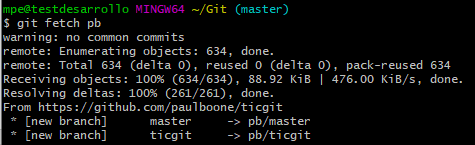
# Vemos que la configuración de los remotos queda grabada en el fichero config

# 

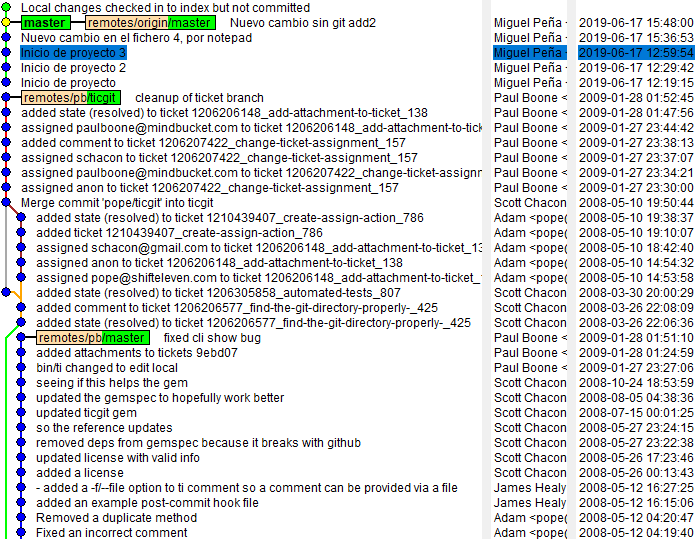


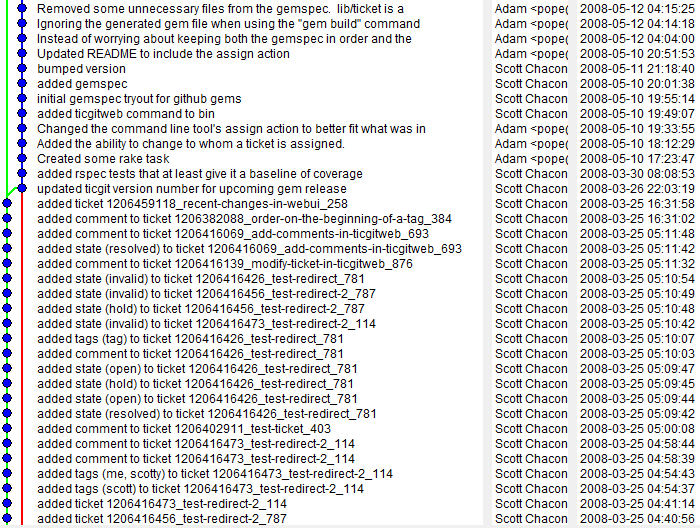
Como vemos podemos usar las abreviaturas origin y pb en lugar de las URL´s para referirnos a los repositorios, por ejemplo, si nos queremos traer el repositorio de Paul ponemos desde línea de comandos

$ git fetch pb



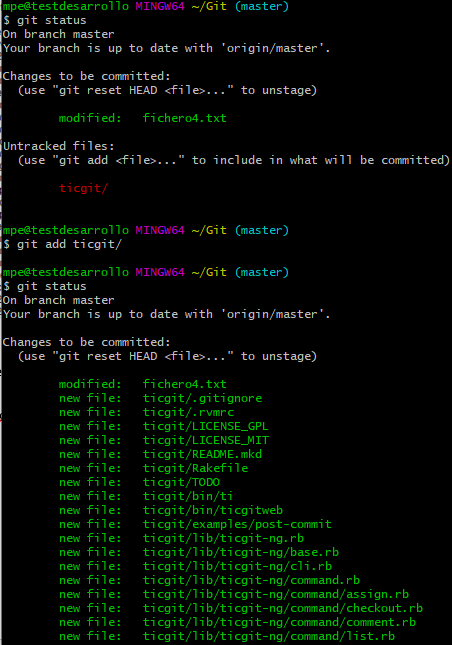
Vemos desde la herramienta gráfica Git GUI como traemos toda la información a nuestro repositorio GIT



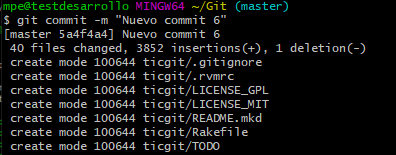


Una vez traído el repositorio de Paul, agregamos todos los archivos a nuestro repositorio local Git y hacemos un commit

$ git add ticgit/



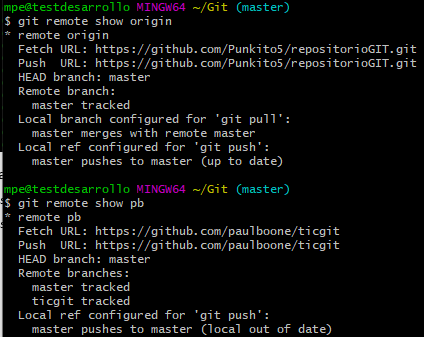
$ git commit -m "Nuevo commit 6"



Para ver información sobre un remoto, tenemos que utilizar el comando show

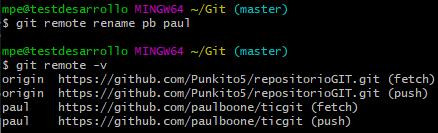
$ git remote show origin

$ git remote show pb



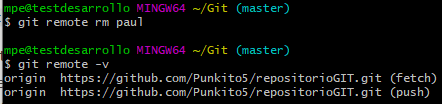
También podemos cambiar las abreviaturas o referencias de los remotos con el comando git remote rename

$ git remote rename pb paul



También se pueden eliminar un remoto de nuestro repositorio GIT con el comando git remote rm

$ git remote rm paul



## Etiquetado

Git utiliza dos tipos principales de etiquetas: ligeras y anotadas.

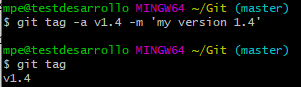
Una etiqueta ligera es muy parecida a una rama que no cambia - simplemente es un puntero a un *commit* específico.

Sin embargo, las etiquetas anotadas se guardan en la base de datos de Git como objetos enteros

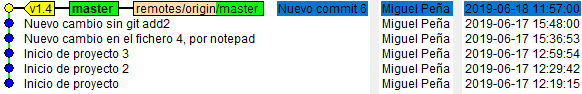
Crear etiquetas anotadas es muy fácil, con el comando git tag y la opción –a

$ git tag -a v1.4 -m 'my version 1.4'

$ git tag



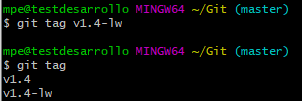
La opción –m es similar a la opción –m del commit, para poner un mensaje en la etiqueta, sino se especifica, se abre el editor que se define en la variable core.editor de la configuración para especificar un mensaje

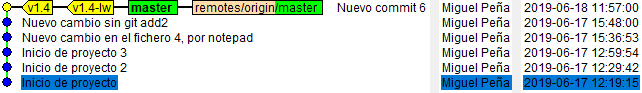


Para crear etiquetas ligeras, sería igual que la creación de las etiquetas anotadas, pero sin la opción –a –m y –s

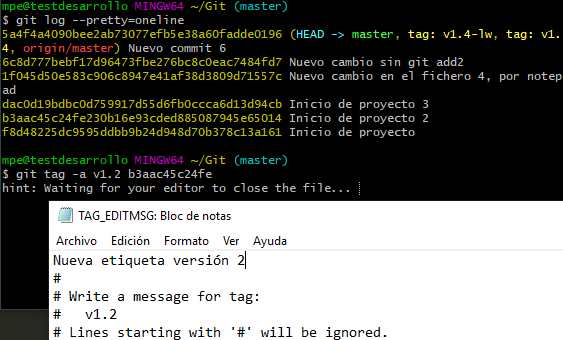
$ git tag v1.4-lw

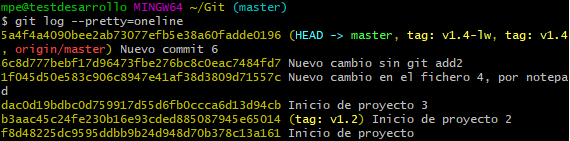
$ git tag

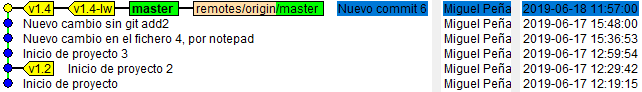




Una cosa interesante que podemos hacer es etiquetar commit donde no pusimos una etiquetas, por ejemplo en el segundo commit “Inicio de proyecto 2” vamos a crear la etiqueta v1.2

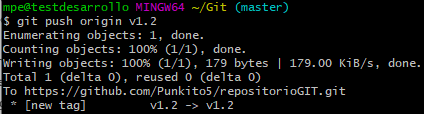




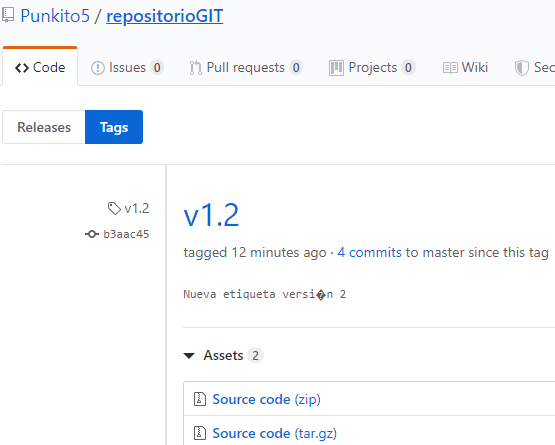


Por defecto, el comando git push no transfiere las etiquetas a los servidores remotos. Debes enviar las etiquetas de forma explícita al servidor luego de que las hayas creado. Este proceso es similar al de compartir ramas remotas - puedes ejecutar git push origin [etiqueta]

$ git push origin v1.2



Comprobamos que esa etiqueta se ha enviado a nuestro servidor remoto



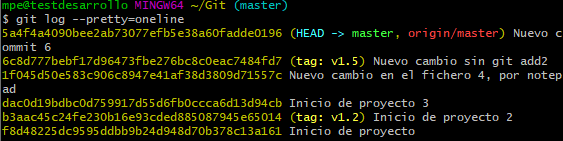
En el source code, tenemos una imagen fidedigna del código que teníamos en la etiqueta v1.2, correspondiente a nuestro segundo commit “Inicio de proyecto 2”

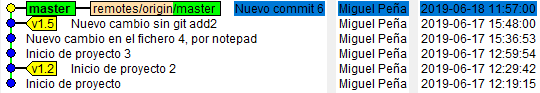
También podemos borrar etiquetas con la opción –d

$ git tag -d v1.4



Borramos las etiquetas v1.4 y v1.4-lw, y creamos la etiqueta v1.5



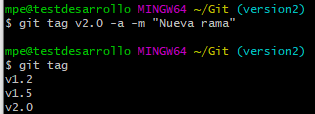


En Git, no puedes sacar (*check out*) una etiqueta, pues no es algo que puedas mover. Si quieres colocar en tu directorio de trabajo una versión de tu repositorio que coincida con alguna etiqueta, debes crear una rama nueva en esa etiqueta

$ git checkout -b version2

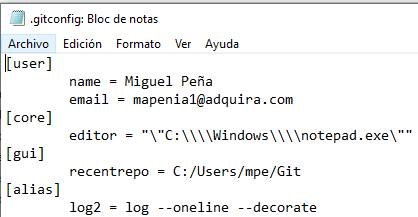
$ git tag v2.0 -a -m "Nueva rama"

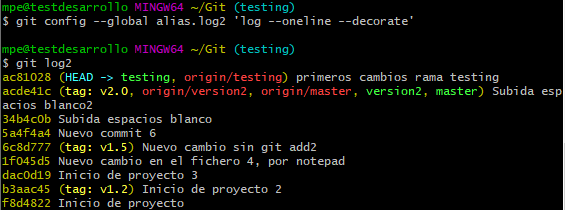




Por último, antes de empezar con las ramas, decir que se pueden definir alias para comando específicos, por ejemplo, vamos a definir un alias para sacar la instrucción git log --oneline –decorate. Se puede hacer mediante comando o escribiendo en el fichero de configuración .gitconfig

$ git config --global alias.log2 'log --oneline --decorate'



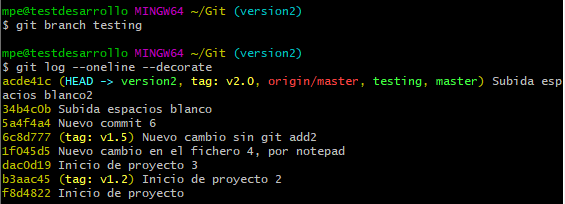


## Ramas

## Cuando hablamos de ramificaciones, significa que tú has tomado la rama principal de desarrollo (master) y a partir de ahí has continuado trabajando sin seguir la rama principal de desarrollo

## Para crear una rama nueva se usa el comando git branch

$ git branch testing



Para saber dónde está actualmente apuntando el apuntador HEAD, podemos ejecutar la sentencia git log

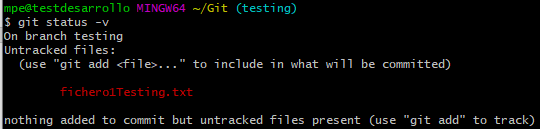
$ git log --oneline --decorate

Para saltar de una rama a otra, tienes que utilizar el comando git checkout. Hagamos una prueba, saltando a la rama testing recién creada:

$ git checkout testing



Ahora nos creamos un nuevo archivo (fichero1Testing.txt) para agregarlo a la nueva rama testing

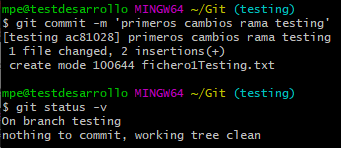


Lo agregamos a nuestro repositorio GIT y hacemos el commit

$ git add fichero1Testing.txt

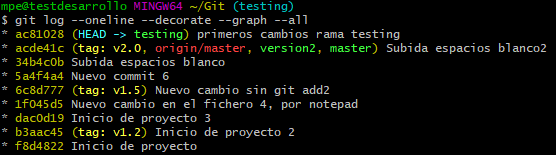


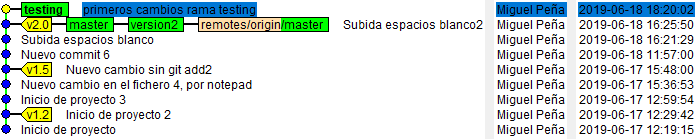
$ git commit -m 'primeros cambios rama testing'



Y vemos que ahora las ramas divergen

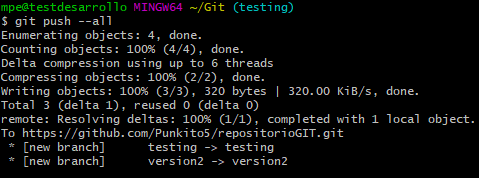
$ git log --oneline --decorate --graph –-all



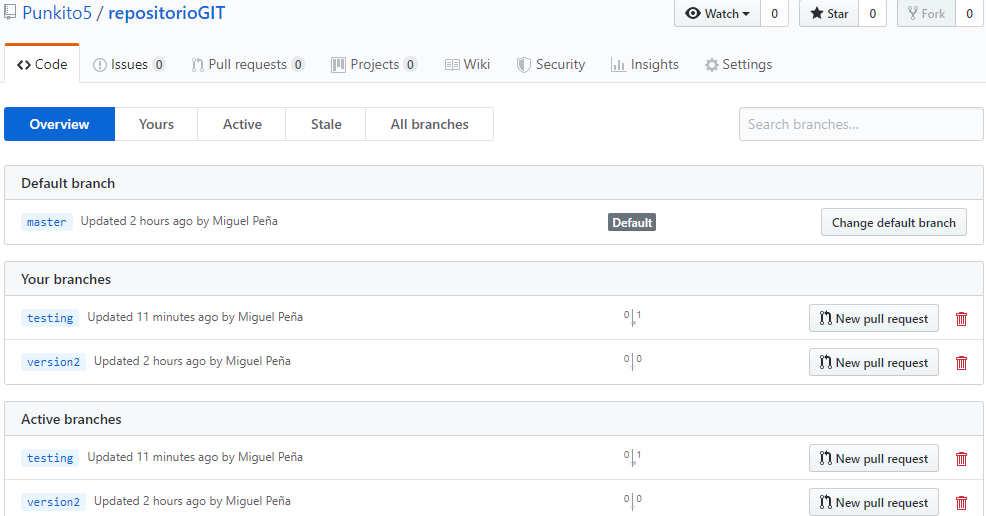


Antes de continuar vamos a proceder a realizar el push a nuestro servidor remoto

$ git push -–all



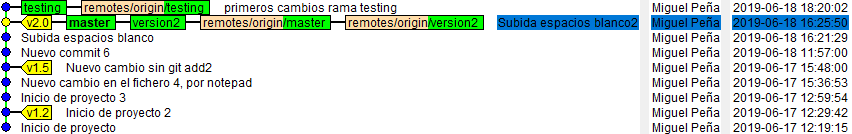
Vemos que tenemos 3 ramas, master, version2 y testing



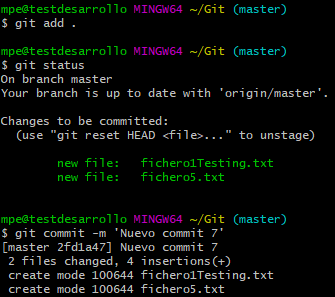
Ahora vamos a cambiarnos a la rama principal de nuevo con el comando git checkout master y vemos que el apuntador HEAD esta en master ahora

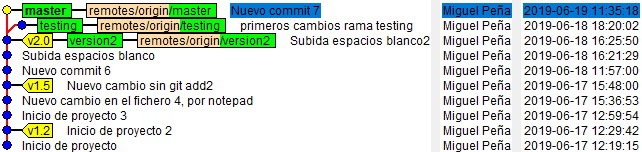
$ git checkout master





Ahora vamos a realizar algún cambio en esta rama como por ejemplo crear otro fichero (fichero5.txt) y lo subiremos con push a nuestro servidor remoto ubicado en github.com viendo que nuestras ramas master y testing divergen

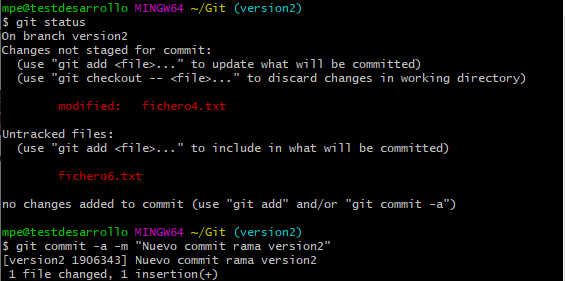




Ahora vamos a saltar a la rama version2 y hacer algunos cambios para posteriormente hacer un merge con la rama principal, la rama master. Una vez estamos en la rama version2, vamos a crear un nuevo archivo fichero6.txt y modificar el fichero4.txt

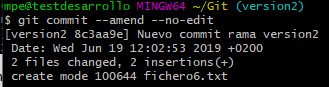


Hacemos el commit de los cambios y cambiamos a la rama master y hacemos un merge con la rama version2

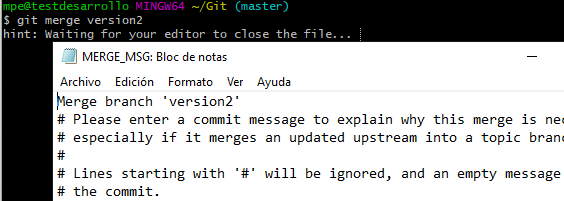


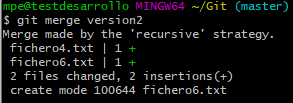
En este caso se me ha olvidado agregar el fichero6.txt al repositorio local Git, así que lo añado y lo agrego al último commit hecho sin modificar el comentario del commit

$ git commit --amend --no-edit

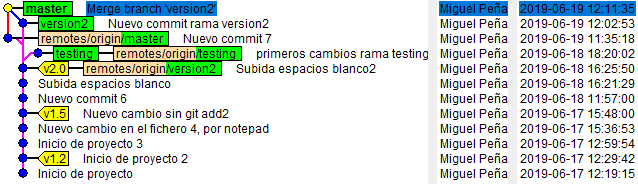


Cambiamos a la rama master y hacemos un merge con la rama version2





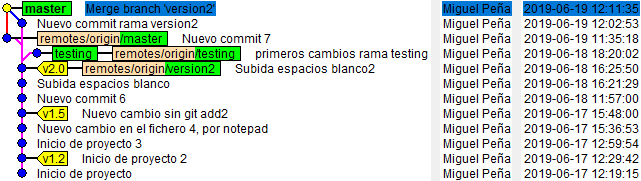
Ahora ya tenemos en nuestra área de trabajo local los cambios del fichero4.txt y el nuevo fichero6.txt que se modificaron en la rama version2



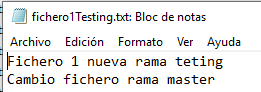
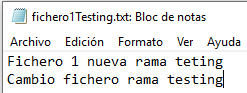
Ahora vamos a proceder a borrar la rama version2, ya que no tiene ningún sentido y todos sus cambios los tenemos centralizados en la rama principal master

$ git branch -d version2





Muy bien, ahora vamos a realizar un cambio en el archivo fichero1Testing.txt que tenemos en la rama master y en la rama testing para ver qué problemas podemos tener a la hora de hacer un merge

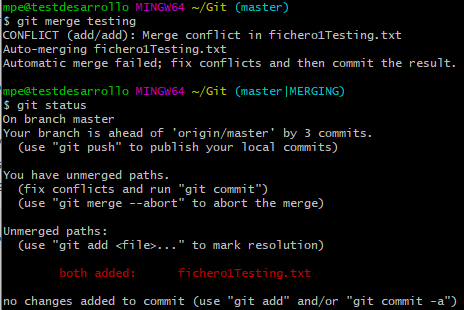
Hacemos un commit en ambas ramas y ahora desde la rama master, hacemos un merge con la rama testing

$ git commit –a -m "Commit merge ficheroTestin master"

$ git commit -a -m "Commit merge ficheroTestin testing"

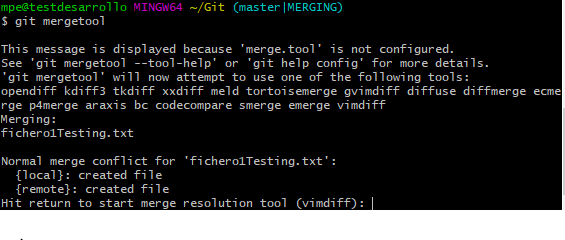


$ git merge testing

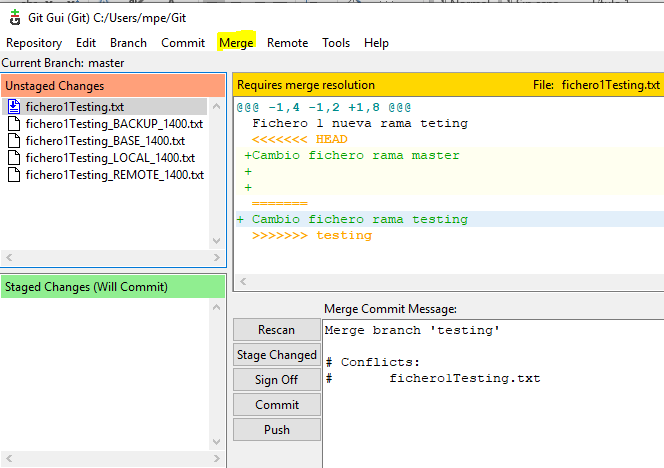


Vemos que hay conflicto en el merge, y para resolverlo podemos lanzar la herramienta mergetool con el comando git mergetool

$ git mergetool



Desde la herramienta gráfica Git GUI, tambien tenemos una opción de merge que es mas sencilla de utilizar



Aunque viendo en internet, una de las mejores herramientas para hacer el merge es la herramienta KDiff3, la cual es gratuita y se puede descargar desde el enlace <https://sourceforge.net/projects/kdiff3/files/>

Una ver la tenemos descargada vamos a configurar nuestro git para que por defecto habrá esta herramienta de comparación cuando hacemos un mergetool

$ git config --global --add merge.tool kdiff3

$ git config --global --add mergetool.kdiff3.path "C:/Program Files/KDiff3/kdiff3.exe"

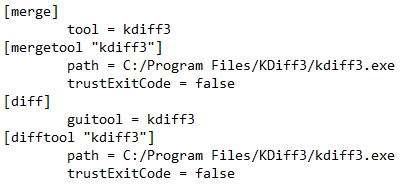
$ git config --global --add mergetool.kdiff3.trustExitCode false

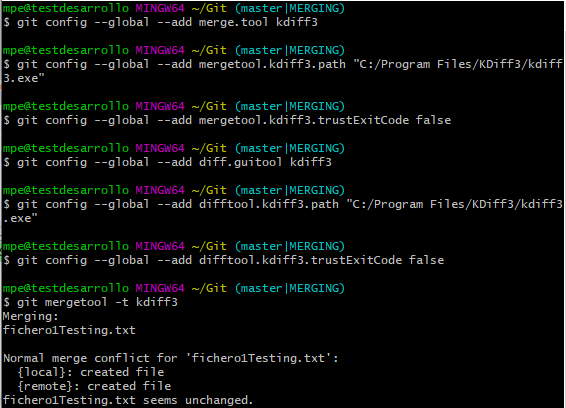
$ git config --global --add diff.guitool kdiff3

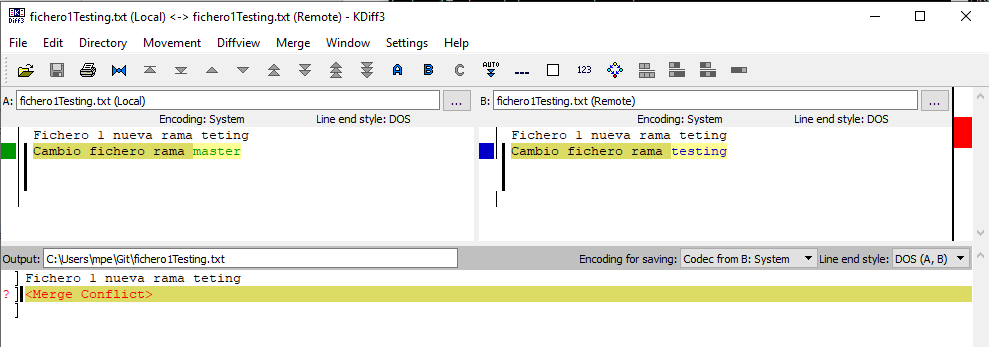
$ git config --global --add difftool.kdiff3.path "C:/Program Files/KDiff3/kdiff3.exe"

$ git config --global --add difftool.kdiff3.trustExitCode false

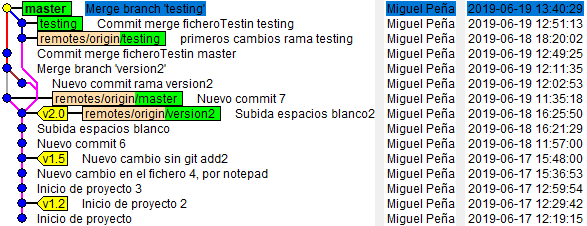
$ git mergetool -t kdiff3







Una vez resuelto los conflictos se hace un commit en la rama master y se hace el merge



## Gestión de Ramas

Lo siguiente que vamos a ver es la gestión de las ramas, para lo que existe el comando git branch, para ver las ramas, cual está activa y los ultimos commit en casa rama

$ git branch –v



Si deseas ver las ramas que han sido fusionadas en la rama activa, puedes lanzar el comando git branch –merged

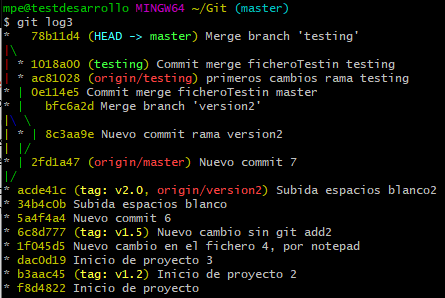


## Ramas Remotas

Las ramas remotas son referencias al estado de las ramas en tus repositorios remotos. Son ramas locales que no puedes mover; se mueven automáticamente cuando estableces comunicaciones en la red. Las ramas remotas funcionan como marcadores, para recordarte en qué estado se encontraban tus repositorios remotos la última vez que conectaste con ellos.

Suelen referenciarse como (remoto)/(rama). Por ejemplo, si quieres saber cómo estaba la rama master en el remoto origin, puedes revisar la rama origin/master

Por ejemplo en nuestro servidor remoto, aun hay tres ramas origin/testing, origin/master y origin/version, cada una apuntando a un commit desde donde se realizó el último push de nuestro servidor Git local al nuestro remoto



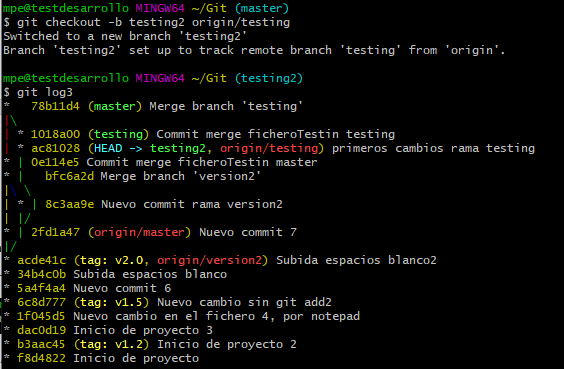
Como vemos en este ejemplo el trabajo local y el trabajo remoto pueden divergir

Para sincronizarte, puedes utilizar el comando git fetch origin



En nuestro caso, por ejemplo, queremos trabajar sobre la rama del remoto origin/testing, para ello lo que tenemos que hacer es crearnos una nueva rama local de seguimiento que trabaje sobre lo que tenía la rama remota origin/testing.

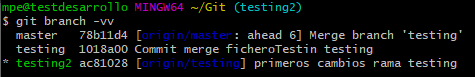
$ git checkout -b testing2 origin/testing



Como vemos, ahora tenemos la copia del código origin/testing que estaba en remoto en nuestro Git local con el nombre de la rama testing2

Si quieres ver las ramas de seguimiento que tienes asignado, puedes usar la opción -vv con git branch. Esto listará tus ramas locales con más información, incluyendo a qué sigue cada rama y si tu rama local está por delante, por detrás o ambas

$ git branch -vv



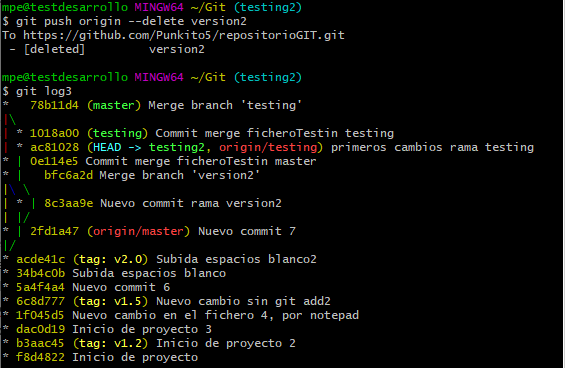
Aquí vemos que nuestra rama master local va por delante de la rama master del servidor remoto y tiene 6 cambios pendientes

A pesar de que el comando git fetch trae todos los cambios del servidor que no tienes, este no modifica tu directorio de trabajo. Simplemente obtendrá los datos y dejará que tú mismo los fusiones. Sin embargo, existe un comando llamado git pull, el cuál básicamente hace git fetch seguido por git merge en la mayoría de los casos

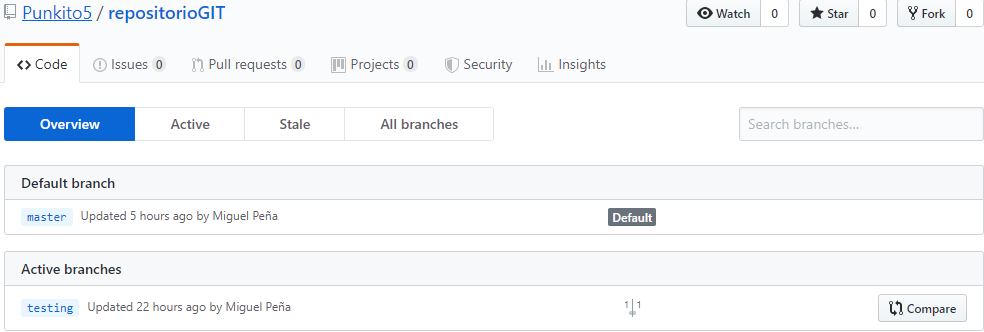
Normalmente es mejor usar los comandos fetch y merge de manera explícita pues la magia de git pull puede resultar confusa.

Otra opción interesante es borrar ramas remotas del servidor remoto. En nuestro caso vamos a borrar la rama origin/version2, que ya borramos localmente

$ git push origin --delete version2

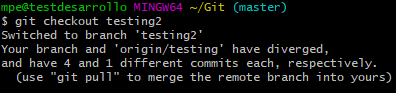


Comprobamos que se borra de nuestro servidor remoto

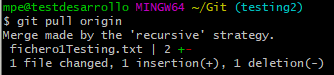


Si por ejemplo nos queremos cambiar de rama y en ella hay cambios que no han sido actualizados en local, el servidor no avisará.

$ git checkout testing2

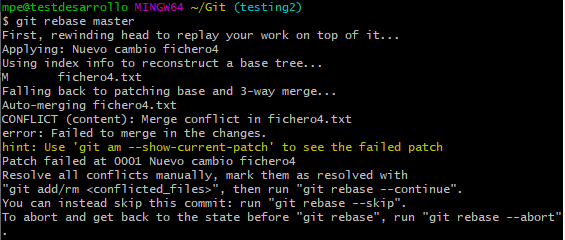


Como vemos tenemos que hacer una actualización.



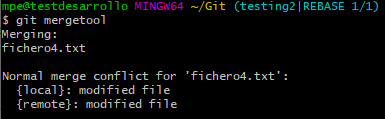
Cuando las ramas divergen, puedes coger todos los cambios confirmados en una rama, y replicarlos sobre otra mediante el comando git rebase

$ git rebase master



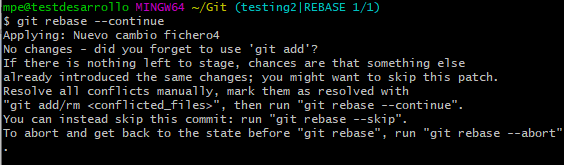
Como vemos resolvemos los conflictos manualmente

$ git mergetool

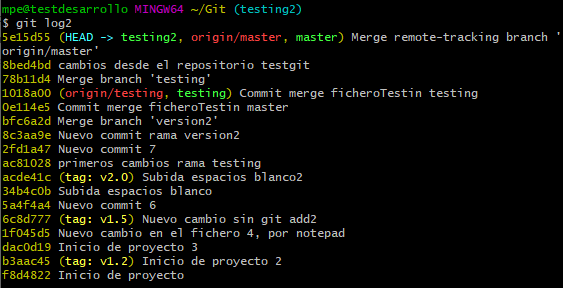


Y continuamos haciendo el rebase

$ git rebase –continue

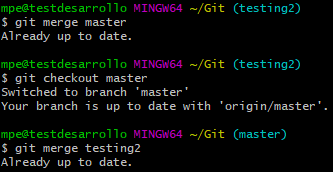


Una vez hecho el rebase, comprobamos que las ramas testing2 y master tienen el mismo contenido

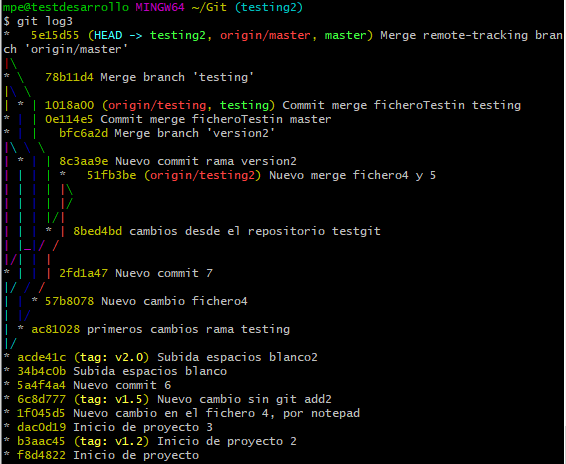


Vemos que el apuntador HEAD está en la rama master y testing concurrentemente

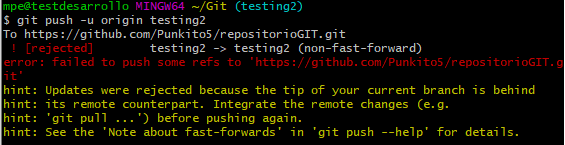
Ahora hacemos un merge de master para comprobar que es el mismo código y viceversa



Ahora lo que queremos hacer es igualar la versión de nuestra rama local testing2, con la rama remota origin/testing2

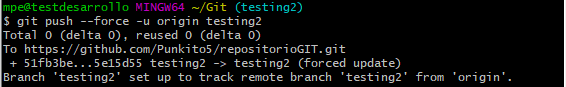


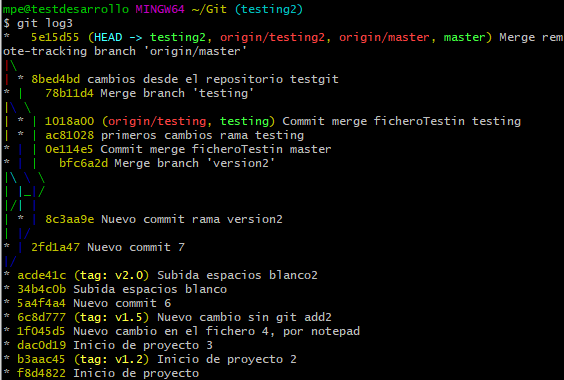
$ git push -u origin testing2



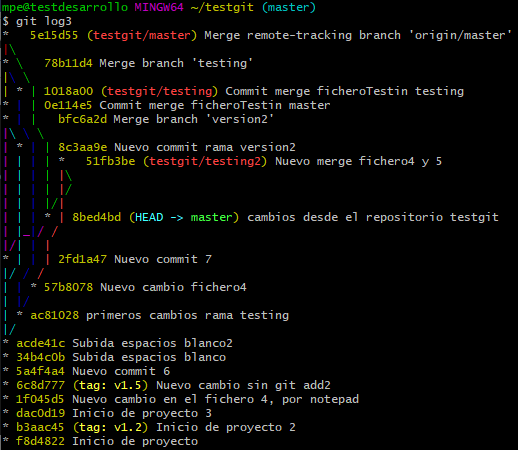
Vemos que hay un conflicto, según el cual no se puede actualizar un push sin avance rápido (fast forwards). Como no he sido capaz de resolver dicho conflicto, se puede forzar el push con la opción –force para igualar las versiones. El problema de esto es que se pierden el histórico de actualizaciones que teníamos

$ git push --force -u origin testing2



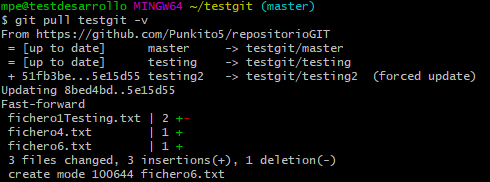


Como tenemos 2 servidores locales, Git y testgit para simular el trabajo de 2 usuarios a la hora de subir cambios a nuestro servidor remoto, en nuestro servidor local testgit, tenemos actualmente el siguiente historial de confirmaciones

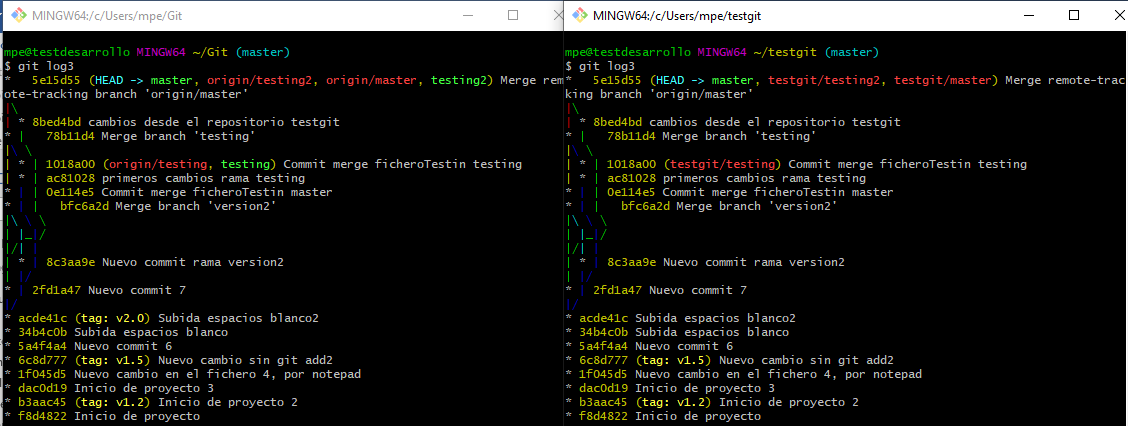


Nos traemos toda la información del servidor remoto

$ git pull testgit –v

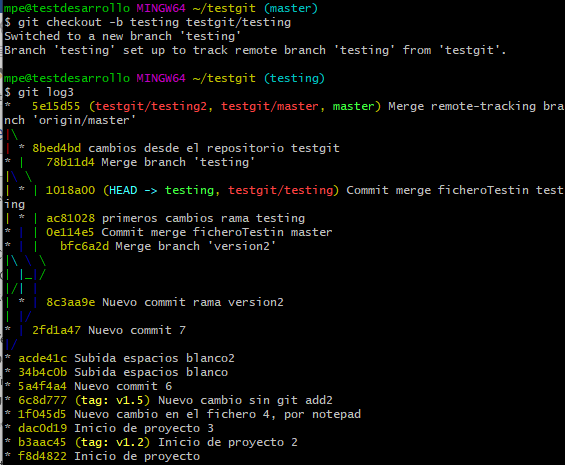


Ahora tenemos la misma información remota tanto en el servidor local Git, como en el servidor local testgit



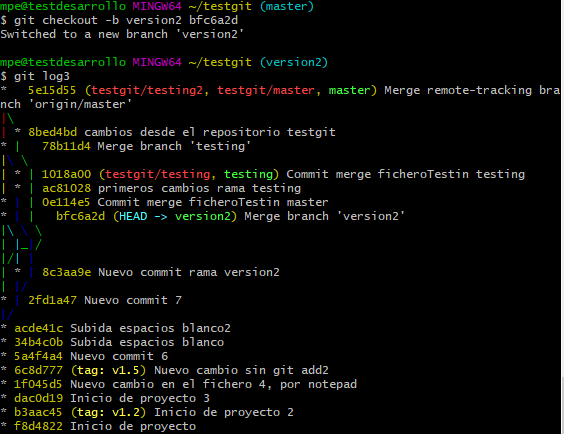
Como vemos, en nuestro servidor local testgit sólo tenemos una rama local, la master. Pero queremos crearnos una rama local llamada testing que apunte a la rama remota testgit/testing

$ git checkout -b testing testgit/testing



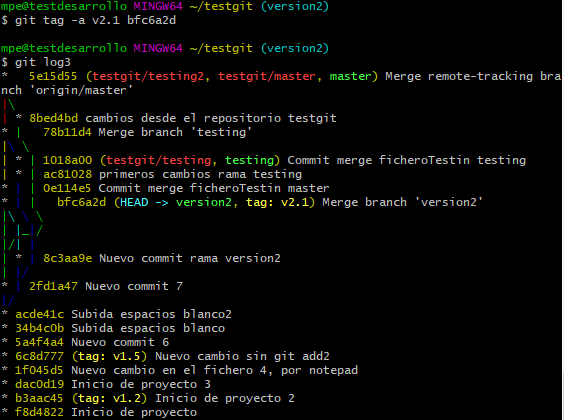
Ahora queremos crearnos otra rama que apunte a un commit determinado, por el ejemplo bfc6a2d Merge branch 'version2'

$ git checkout -b version2 bfc6a2d



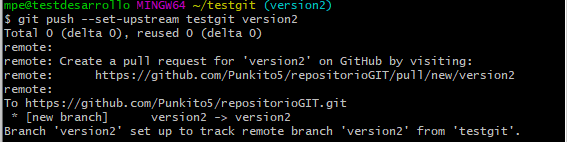
Aparte de la rama, vamos a crear una etiqueta v2.0 y subirla al servidor remoto para tener una reléase en ese punto

$ git tag -a v2.1 bfc6a2d



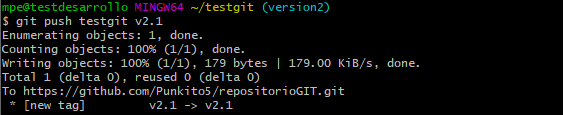
Subimos la nueva rama al servidor remoto

$ git push --set-upstream testgit version2



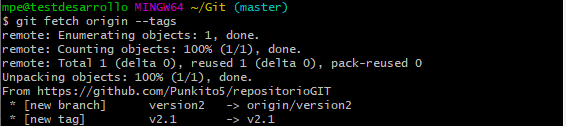
Y la etiqueta

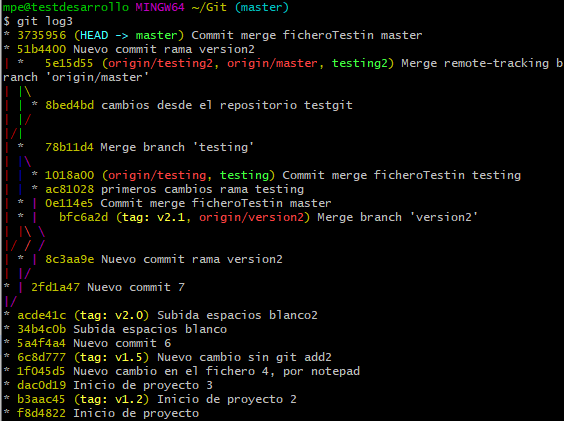
$ git push testgit v2.1



Ahora nos movemos a nuestro servidor local Git y nos traemos los últimos cambios, es decir la etiqueta v2.1 y la nueva rama origin/version2

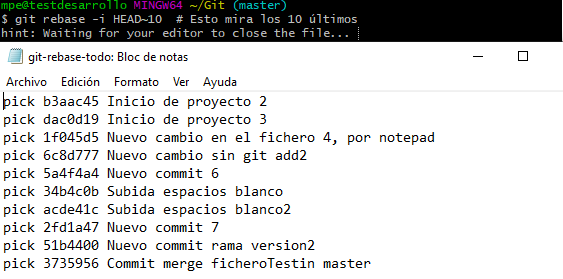
$ git fetch origin –tags

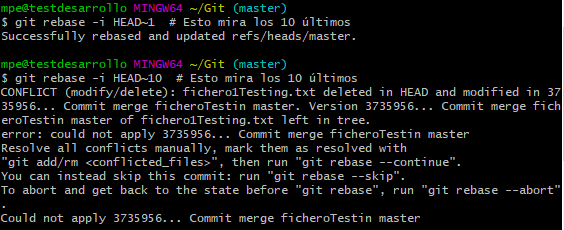




Ahora vamos a aprender como borrar commit aleatoriamente. En el repositorio local Git nos movemos a la rama master y borramos el commit pick 2fd1a47 Nuevo commit 7

$ git rebase -i HEAD~10 # Esto mira los 10 últimos



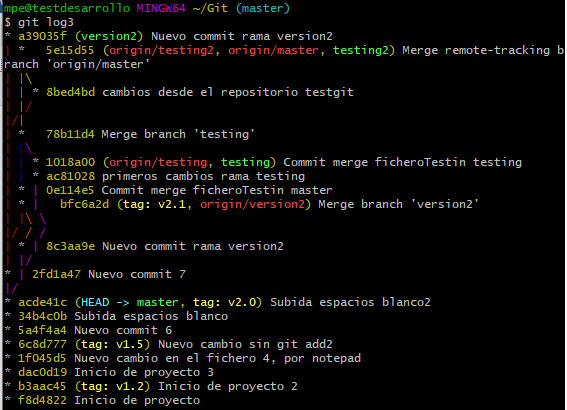


Si queremos borrar el ultimo commit hacemos lo siguiente

$ git reset HEAD

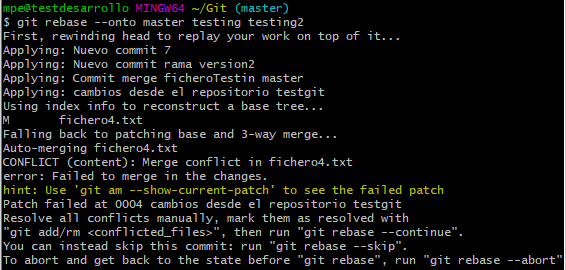


Vemos que tenemos 4 ramas locales, y que la rama master, donde hemos procedido a borrar unos cuantos commits, está actualmente en el commit Subida espacios blanco2

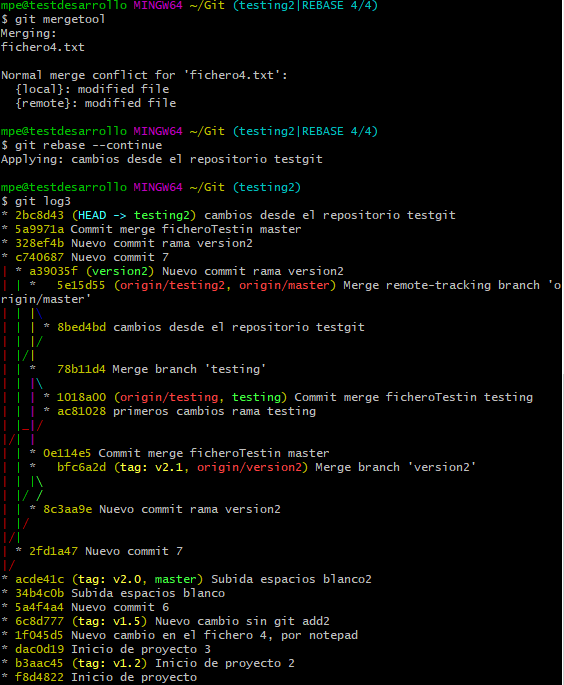


Vamos a intentar hacer un rebase de las demás ramas testing2 y testing a la rama master. Esto viene a decir: “Activa la rama testing, averigua los cambios desde el ancestro común entre las ramas testing y testing2 y aplícalos en la rama master”

$ git rebase --onto master testing testing2

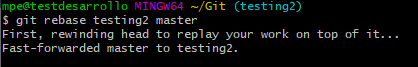


Resolvemos los conflictos y el historial de confirmaciones queda así



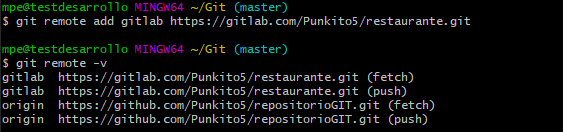
Ahora igualamos las ramas testing2 y master

$ git rebase testing2 master



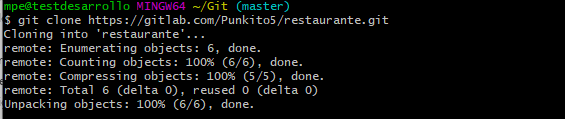
## GitLab

$ git remote add gitlab <https://gitlab.com/Punkito5/restaurante.git>



Ahora hacemos una copia local del repositorio de GitLab

$ git clone <https://gitlab.com/Punkito5/restaurante.git>

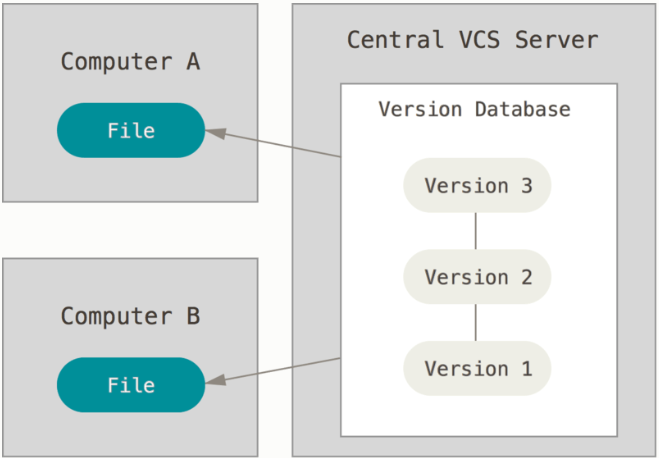


## Flujos de trabajo distribuidos

## La naturaleza distribuida de Git te permite mucha más flexibilidad en la manera de colaborar en proyectos

### Flujos de trabajo centralizado

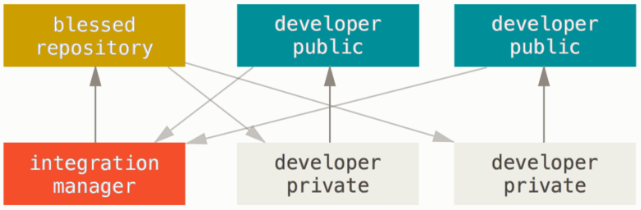
Un repositorio o punto central que acepta código y todos sincronizan su trabajo con él



### Flujo de Trabajo Administrador-Integración

Flujo de trabajo donde cada desarrollador tenga acceso de escritura a su propio repositorio público y acceso de lectura a todos los demás

1. El administrador del proyecto hace un push al repositorio público.
2. El contribuidor clona ese repositorio y realiza los cambios.
3. El contribuidor realiza un push con su copia pública del proyecto.
4. El contribuidor envía un correo electrónico al administrador pidiendo que haga pull de los cambios.
5. El administrador agrega el repositorio del contribuidor como remoto y fusiona ambos localmente.
6. El administrador realiza un push con la fusión del código al repositorio principal.



### Flujo de Trabajo Dictador-Tenientes

1. Los desarrolladores trabajan en su propia rama especifica y fusionan su código en la rama master, la cual, es una copia de la rama del dictador.
2. Los tenientes fusionan el código de las ramas master de los desarrolladores en sus ramas master de tenientes.
3. El dictador fusiona la rama master de los tenientes a su rama master de dictador.
4. El dictador hace push del contenido de su rama master al repositorio para que otros fusionen los cambios a sus ramas.

