GIT

GitHub y manejo básico (III)

Guillermo Palazón Cano

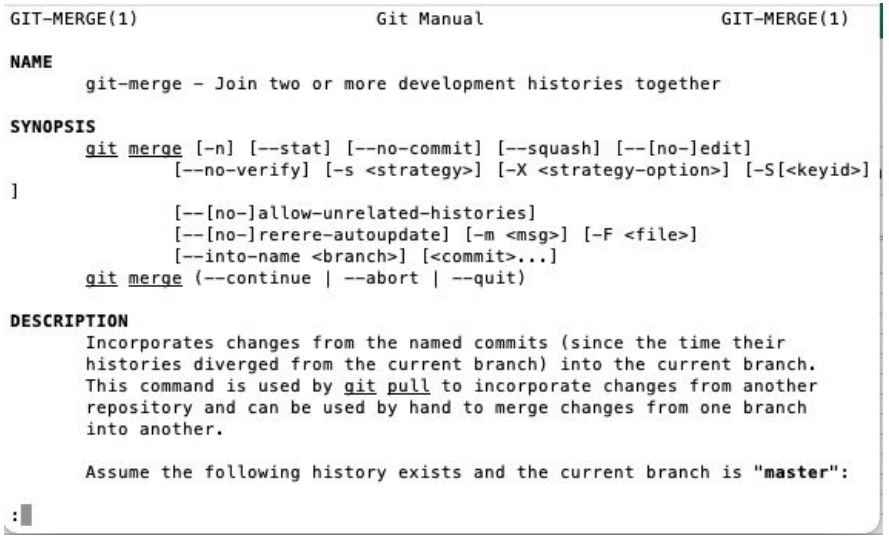
Introducción

- Parte de este material se lo debemos a:
  - Cuaderno de J.R. Johansson (jrjohansson@gmail.com) disponible en http://github.com/jrjohansson/scientific-python-lectures y del libro Pro Git: https://git-scm.com/book/es/v2
  - Otros cuadernos del mismo autor están disponibles en http://jrjohansson.github.io.
  - Adaptación de estos cuadernos realizada por Alberto Sierra Olmo, profesor de la especialidad de Informática en la Comunidad Autónoma de la Región de Murcia.

Breves anotaciones

- Si alguna vez necesitas ayuda usando Git, podemos ver su página del manual (manpage) para cualquier comando de Git de tres formas distintas con similar resultado
  - **git help comando**
  - **git comando –help**
  - **man git -comando**

```
git help merge
git merge --help
man git-merge
```



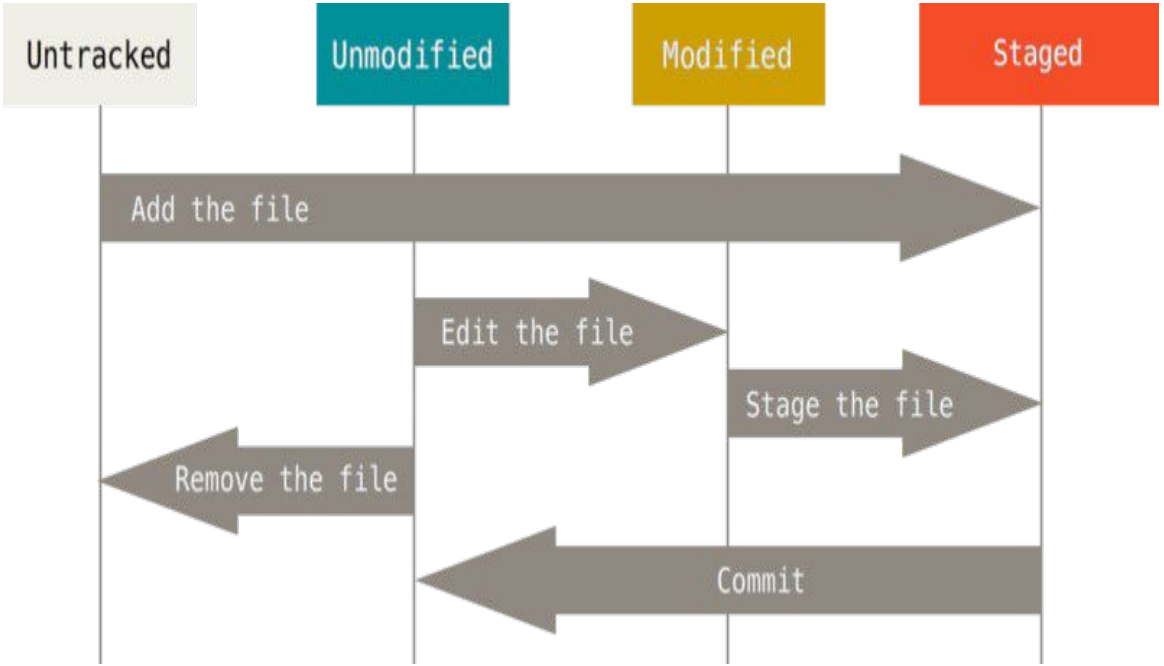
```
GIT-MERGE(1)                    Git Manual                    GIT-MERGE(1)

NAME
       git-merge - Join two or more development histories together

SYNOPSIS
       git merge [-n] [--stat] [--no-commit] [--squash] [--[no-]edit]
               [--no-verify] [-s <strategy>] [-X <strategy-option>] [-S[<keyid>]
]
               [--[no-]allow-unrelated-histories]
               [--[no-]rerere-autoupdate] [-m <msg>] [-F <file>]
               [--into-name <branch>] [<commit>...]
       git merge (--continue | --abort | --quit)

DESCRIPTION
       Incorporates changes from the named commits (since the time their
       histories diverged from the current branch) into the current branch.
       This command is used by git pull to incorporate changes from another
       repository and can be used by hand to merge changes from one branch
       into another.

       Assume the following history exists and the current branch is "master":

:
```

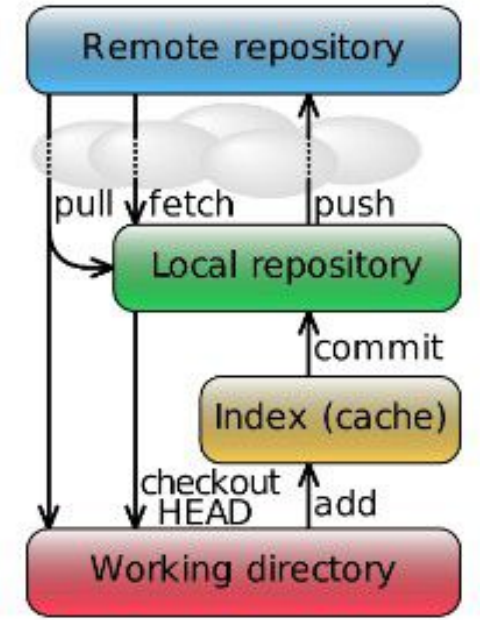
- Conceptos de master (main), origin y HEAD
  - **master** → También se puede encontrar con main. Es la rama predeterminada y se crea automáticamente al crear un repositorio. Se le considera la rama principal y generalmente es la raíz de la mayoría de las demás ramas, por lo que es habitual que en ella se encuentre el "código definitivo", que luego va a producción, y es la rama en la que se mezclan todas las demás más tarde o temprano para dar por finalizada una tarea e incorporarla al producto final.
  - **origin** → Lo más habitual es que nuestro repositorio local trabaje respecto de un (puede ser más de uno) repositorio "remoto". En este caso origin es simplemente el nombre predeterminado que recibe el repositorio remoto principal contra el que trabajamos. Este nombre se puede cambiar y se pueden crear más repositorios remotos.
  - **HEAD** → El concepto de HEAD es muy simple: se refiere al commit (isntantánea) en el que está tu repositorio posicionado en cada momento. Por regla general HEAD suele coincidir con el último commit de la rama en la que estés, ya que habitualmente estás trabajando en lo último. Pero si te mueves hacia cualquier otro commit anterior entonces el HEAD estará más atrás. Si tienes el repositorio actualizado y estás trabajando en la rama master (main) lo más habitual es que coincidan las tres cosas.

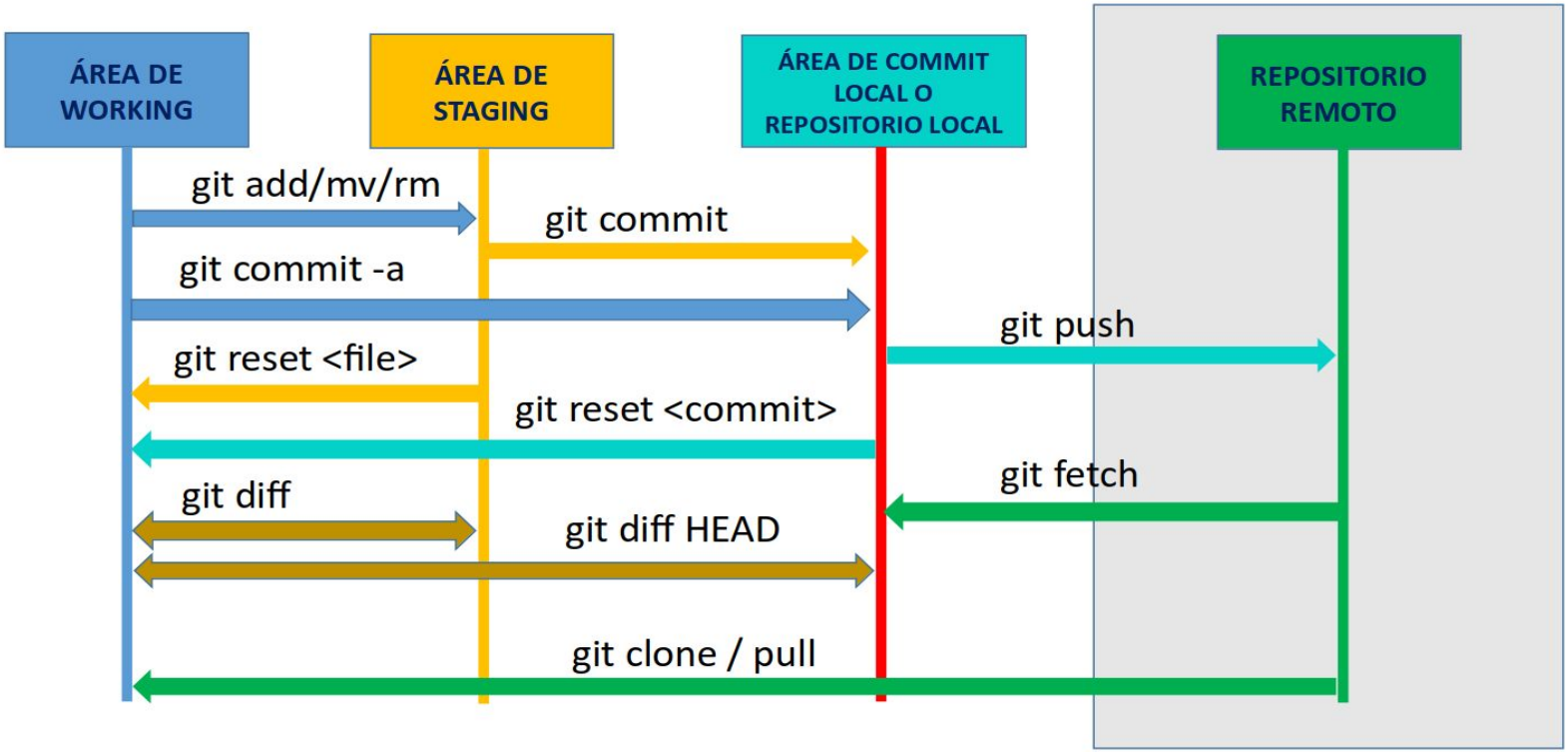
Recordatorio estados archivos

Cada archivo de repositorio puede tener dos estados de manera general:

- Rastreados (tracked) → Son aquellos archivos que estaban en la última instantánea del repositorio o están en el área de preparación
  - Sin modificar
  - Modificados
  - Preparados
- Sin rastrear (untracked) → El resto de los archivos del directorio de trabajo.


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit


Remote repository
pull
fetch
push
Local repository
commit
Index (cache)
checkout HEAD
add
Working directory


ÁREA DE WORKING
ÁREA DE STAGING
ÁREA DE COMMIT LOCAL O REPOSITORIO LOCAL
REPOSITORIO REMOTO
git add/mv/rm
git commit
git commit -a
git push
git reset <file>
git reset <commit>
git fetch
git diff
git diff HEAD
git clone / pull

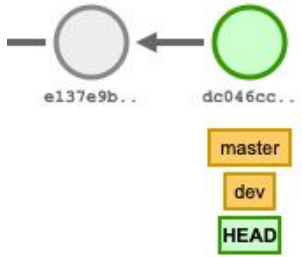
Repaso manejo de ramas

- Vamos a utilizar una herramienta que nos va a permitir ver gráficamente los cambios que se producen en el repositorio según vamos trabajando con ramas.
- Para ellos trabajaremos con un terminal negro donde introducir los comandos de git en la siguiente web: https://onlywei.github.io/explain-git-with-d3/#branch (es habitual en ocasiones que tarde en realizar la carga de la terminal)
- Vamos a ir ejecutando una serie de comandos para revisar el concepto de manejo de ramas en git.
- Esta herramienta NO tiene en cuenta los conflictos que se podrían suceder, hace los merge como si nunca se fueran a producir estos.

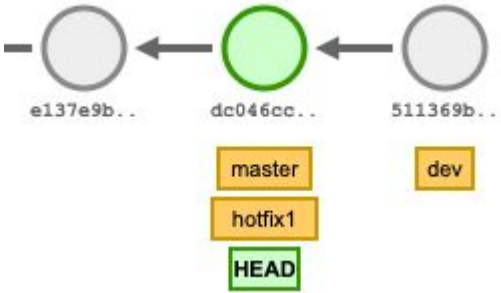
1. **Antes de iniciar.** Podemos ver que tanto HEAD (instantánea en la que está posicionada mi repositorio) como master están posicionados en la instantánea del repositorio con código e137e…

2. **git commit**. Crea una nueva instantánea del repositorio en la rama master (no existe otra). Tanto HEAD como master están posicionado en dicha instantánea

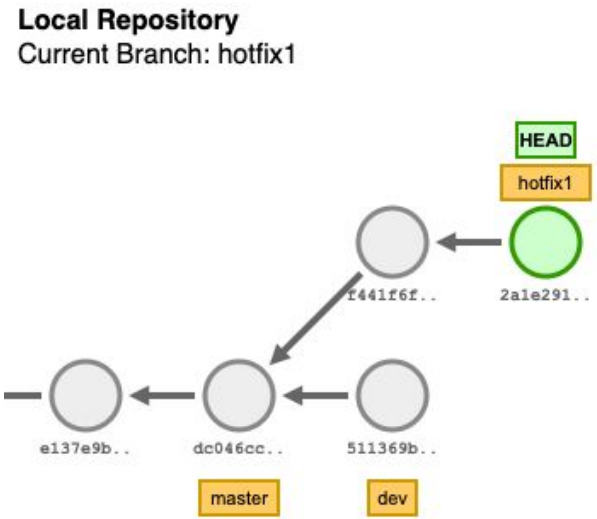
3. **git checkout -b dev**. Este comando es similar a crear una rama dev (git branch dev) y a continuación ir a ella (git checkout dev). Al crear la rama y hacer el checkout ahora el HEAD está posicionado en dev.



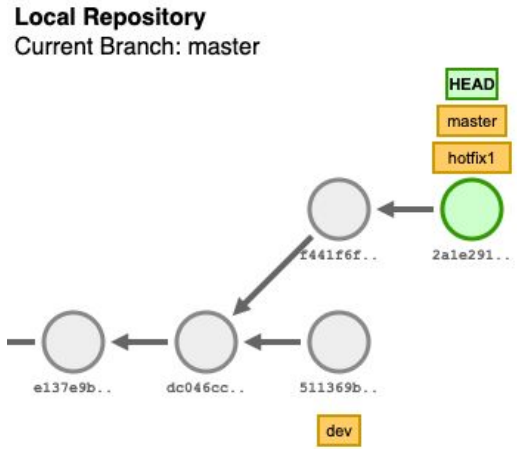
4. **git commit.** Crea una instantánea en la rama dev. HEAD ahora mismo está posicionada en dicha instantánea.
5. **git checkout master.** HEAD ahora mismo está posicionado en la instantánea en la que está posicionada la rama master.
6. **git checkout -b hotfix1**. Se crea una nueva rama hotfix1 a partir de la rama master que es en la que estamos posicionados y HEAD se posiciona con ella.



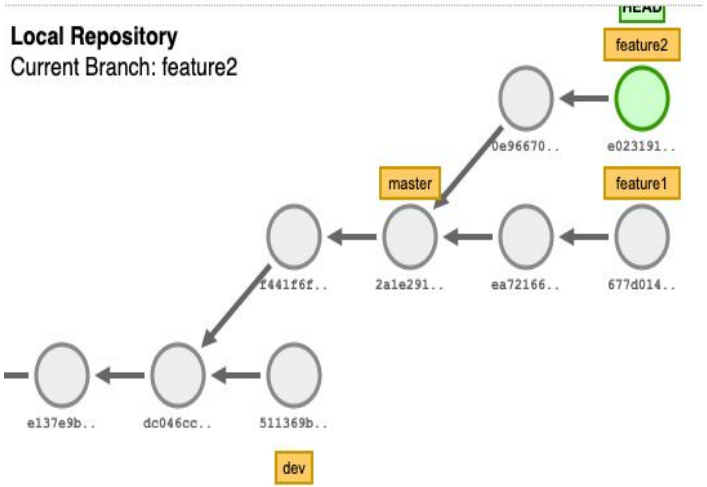
7. **git commit**
8. **git commit.** La rama hotfix1 ha creado dos nuevas instantáneas y HEAD está posicionado en la última que se ha creado.



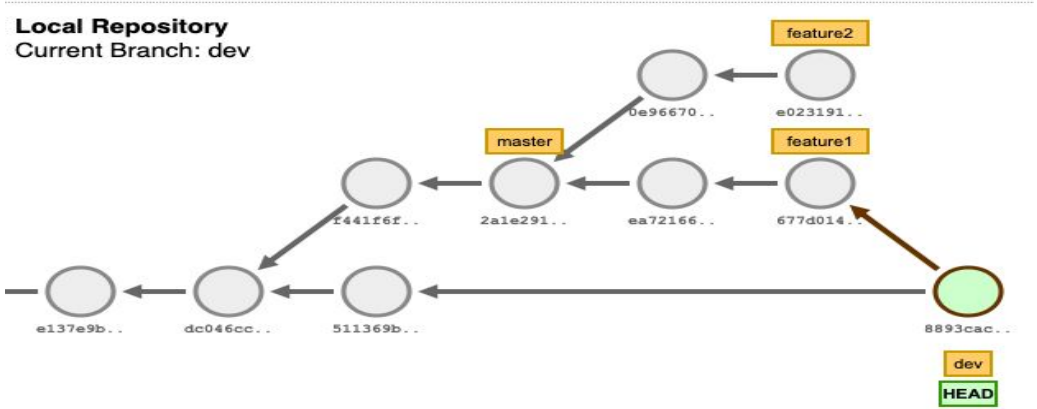
9. **git checkout master.** HEAD se posiciona ahora en la rama master
10. **git merge hotfix1.** Se ha hecho la fusión desde la rama master con la rama hotfix1. Puesto que la rama hotfix1 estaba más “adelantada” y la rama master no había realizado commit desde que se había creado la rama hotfix1, se ha “solucionado” con master posicionado en la instantánea creada por hotfix1 y HEAD apuntando a master. Podemos ver en la terminal que indica “You have performed a fast-forward merge”, es decir, es una fusión de avance rápido



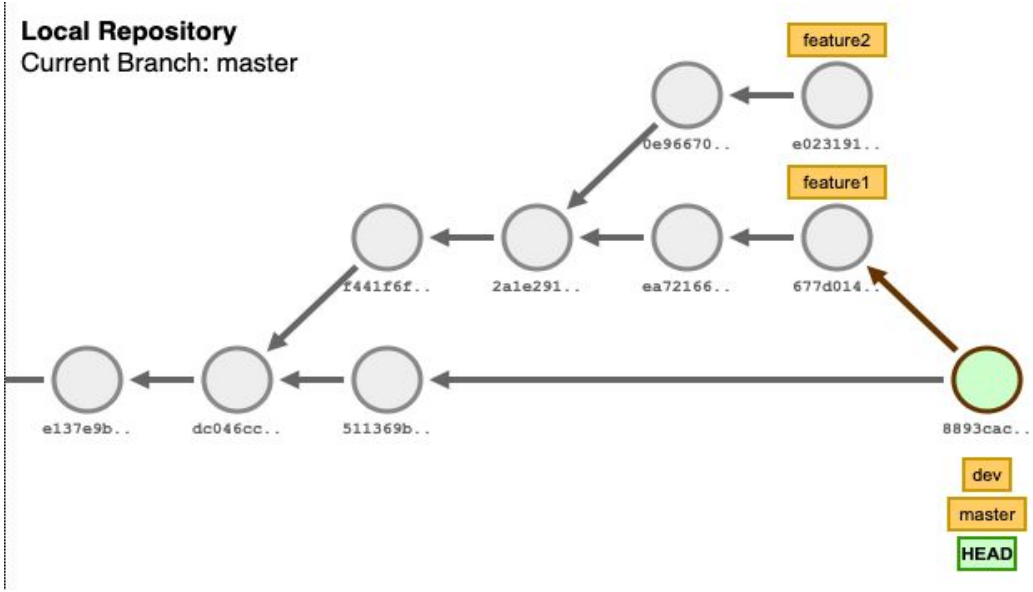
11. **git branch -d hotfix1.** Elimina la rama.
12. **git branch feature1**
13. **git branch feature2.** Crea dos nuevas rama, pero HEAD sigue posicionado en el master.
14. **git checkout feature1**. El HEAD se posiciona en feature1
15. **git commit**
16. **git commit**
17. **git checkout feature2**
18. **git commit**
19. **git commit.** Tanto en la rama feature1 como feature2 se han creado dos instantáneas. En la actualidad HEAD está posicionado en la rama feature2.



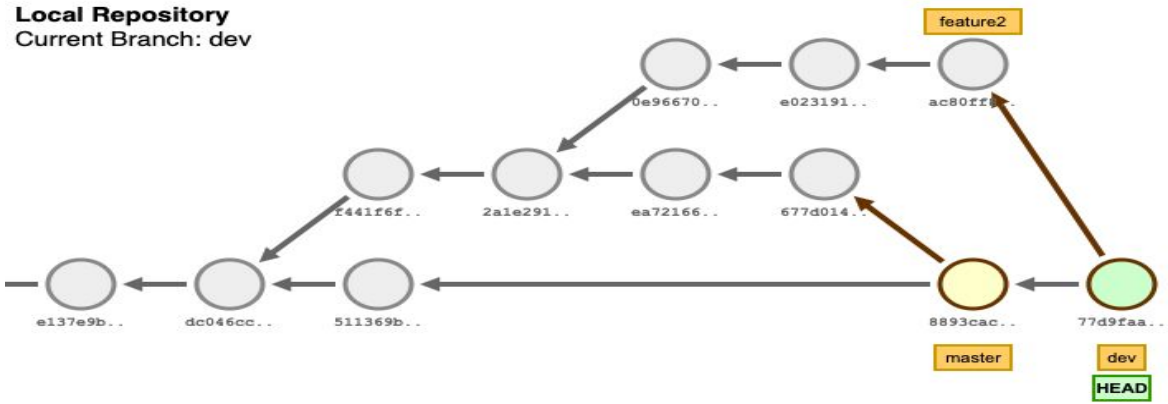
20. **git checkout dev.** Cambiamos a la rama dev, por lo que ahora HEAD está posicionado en la instantánea 511…
21. **git merge feature1**. Se crea una nueva instantánea en la rama dev que incorpora los cambios que se hayan realizado en feature1. HEAD apunta a esta instantánea.



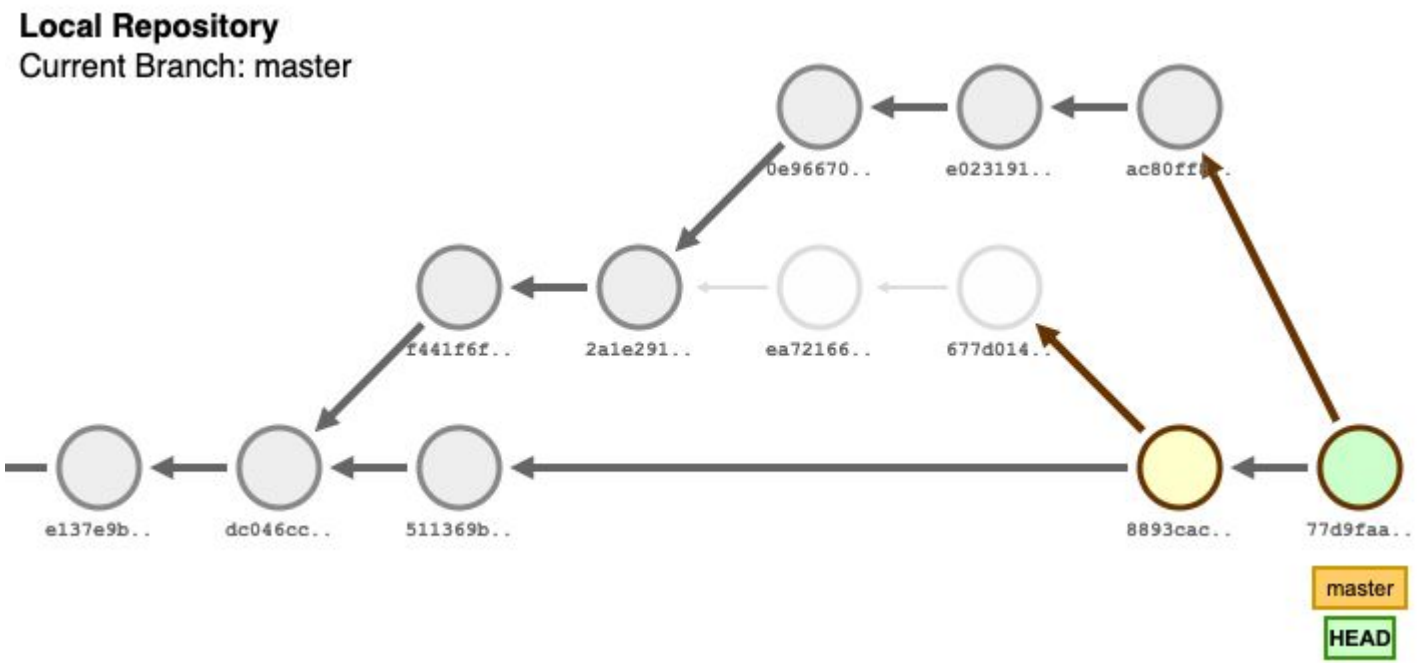
22. **git checkout master.** HEAD se posiciona con master
23. **git merge dev.** Vuelve a realizar una fusión de avance rápido y la instantánea más adelantada de master es 8893…



24. **git branch -d feature1.** Elimina la rama
25. **git checkout feature2.** Cambiamos de rama
26. **git commit**
27. **git checkout dev**
28. **git merge feature2.** Crea una nueva instantánea en dev con los cambios que se tenían en feature2. Esta rama (dev) se adelanta otra vez respecto a master.



29. **git checkout master**
30. **git merge dev.** Vuelve a hacer una fusión de avance rápido que básicamente consiste en avanzar master a la instantánea actual de dev.
31. **git branch -d feature2**
32. **git branch -d dev.** Dejamos master como única rama, la cual está conectada con todas las instantáneas que fueron creando cada una de las ramas a lo largo de los diferentes commit que se han ido produciendo en ellas. HEAD está posicionado en la última instantánea (al igual que master)


Local Repository
Current Branch: master
0e96670..
e023191..
ac80ff..
f441f6f..
2a1e291..
ea72166..
677d014..
e137e9b..
dc046cc..
511369b..
8893cac..
77d9faa..
master
HEAD

Ejemplo/Práctica

- Vamos a clonar un repositorio para realizar poder seguir un ejemplo en el que vamos a repasar cuestiones ya vistas y nuevas que que se van a ir incorporando.
- A lo largo del desarrollo de este ejercicio os voy a ir pidiendo realizar capturas de pantalla que deberéis incorporar a un documento para su posterior entrega.
- Para clonar un repositorio solamente necesitamos conocer su URL pública.
- Vamos a realizar el git clone lo que nos va a crear una carpeta para el repositorio (en mi caso he creado el repositorio dentro de una carpeta PruebasGit pero no sería necesario realmente).
- La dirección del repositorio es (**voy a utilizar en algunas capturas OTRO repositorio DIFERENTE para no modificar el que se utiliza en la práctica, lo aviso porque algunas capturas podrán diferir**): https://github.com/guillermopalazon/practicaJuegoED20212022.git
- Realizar el git clone del mismo como se puede realizar en la siguiente diapositiva (Crea captura de pantalla 01)

```
MacBook-Air-de-Guillermo:PruebasGit guillermopalazoncano$ git clone https://github.com/guillermopal
azon/practicaJuegoED20212022.git
Clonando en 'practicaJuegoED20212022'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 23 (delta 3), reused 0 (delta 0), pack-reused 0
Recibiendo objetos: 100% (23/23), 65.08 KiB | 550.00 KiB/s, listo.
Resolviendo deltas: 100% (3/3), listo.
```

- Al clonar el repositorio se crea automáticamente una conexión con el repositorio remoto que tendrá el nombre por defecto de origin
- Se pueden ver todas las conexiones con repositorios remotos que tenemos en un repositorio (ya que se pueden añadir otras con el comando ya visto git remote add) con el comando git remote sin parámetros.
- Podemos obtener más información de una conexión remota con el comando git remote show nombre_conexion
- Muestra esta información en tu repositorio (Captura de pantalla 02)

```
MacBook-Air-de-Guillermo:practicaJuegoED20212022 guillermopalazoncano$ git remote
origin
MacBook-Air-de-Guillermo:practicaJuegoED20212022 guillermopalazoncano$ git remote show origin
* remoto origin
  URL  para obtener: https://github.com/guillermopalazon/practicaJuegoED20212022.git
  URL para publicar: https://github.com/guillermopalazon/practicaJuegoED20212022.git
  Rama HEAD: main
  Rama remota:
    main rastreada
  Rama local configurada para 'git pull':
    main fusiona con remoto main
  Referencia local configurada para 'git push':
    main publica a main (actualizado)
```

- Podemos recibir actualizaciones, ejecutando un **git pull** desde el repositorio **origin** remoto hacia nuestro repositorio local: git pull origin (o simplemente: git pull)
- Podemos tener registradas direcciones hacia distintos repositorios, y obtener (pull) las actualizaciones desde diferentes fuentes, pero por defecto la fuente es el origen desde donde se clonó inicialmente el repositorio.

- Tras realizar cambios en nuestro repositorio local, podemos enviarlos al repositorio remoto, utilizando **git push**. De nuevo, el repositorio de destino por defecto es origin.
- Tienes que añadir un nuevo repositorio de que debe tener el siguiente nombre: nombreprueba, donde nombre es tu nombre. Por ejemplo, en mi caso sería guillermoprueba.
- Este archivo puede estar vacío, por lo que puedes crearlo por ejemplo con el comando touch guillermoprueba (en algún sistema operativo es posible que no exista este comando y debas crearlo de otra forma)
- Realiza la subida de este fichero al repositorio remoto (al github). Para realizar push en un repositorio que no has creado tú, tienes que tener los permisos adecuados concedidos por el propietario del repositorio remoto. Como propietario te he invitado a ser colaborador del repositorio para tu cuenta @alu.murcia el repositorio (deberás aceptar la invitación a ser colaborador antes de hacer el push, sino te dará error de permisos). En caso de no tener invitación o usar otra cuenta diferente deberás ponerte en contacto conmigo.
- MUY IMPORTANTE: Es posible que tengas que hacer un git pull antes de hacer el git push que verás en la siguiente pantalla si algún compañero ha realizado un git push desde el momento en el que tú realizaste el git pull → git pull origin main --rebase

- NOTA: En el repositorio la rama principal no se llama master sino main.
- Una vez creado el archivo y añadido a tu repositorio local se deberá realizar el git push (Captura de pantalla 03)

```
MacBook-Air-de-Guillermo:practicaJuegoED20212022 guillermopalazoncano$ git push origin main
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 315 bytes | 315.00 KiB/s, listo.
Total 3 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/guillermopalazon/practicaJuegoED20212022.git
   4101470..75ea3d6  main -> main
```

- A continuación nos vamos a traer las modificaciones que existen en nuestro repositorio, que serán las subidas de ficheros vacíos que hayan hecho los compañeros antes de realizar este comando.
- El número de archivos que obtendremos variará en función de la premura en la que haya realizado los apartados anteriores. Si soy el primero en llegar a este punto, únicamente me va a descargar el fichero vacío realizado por el profesor (en mi caso como es evidente NO va a descargar nada)
- Realiza el git pull origin main --rebase (Captura de pantalla 04)
- Realiza un git log (Captura de pantalla 05). Además de adjuntar esta captura de pantalla deberás comentar brevemente la misma

FORK

- Cuando hacemos un fork de un repositorio de github (en internet), se hace una copia exacta del repositorio original, del que seguramente no tenemos derechos de escritura, almacenándose la copia en nuestra propia cuenta de github (en internet) y lo podemos utilizar como si fuera nuestro propio repositorio, con derechos de escritura.
- Para hacer un fork de un repositorio nos colocamos dentro del mismo en GitHub y en el caso que lo permita se deberá buscar la opción Fork y rellenar el formulario que saldrá a continuación

- Realiza el Fork del repositorio con el que se está trabajando hasta ahora (Captura de pantalla 06)

PULL REQUEST

- Un pull request es una característica particular de algunas de las plataformas web que implementan Git, como Github o Bitbucket.
- Es una solicitud para integrar una rama en el proyecto (es decir, realizar un merge en github en internet, no en local)
- El objetivo final de un pull request es realizar una integración (merge) de una rama que incluye una serie de modificaciones, en el proyecto para el que han sido desarrolladas.
- Un pull request posibilita la comunicación entre los miembros del equipo mediante notificaciones, comentarios y revisiones de código, dado lugar a un foro de discusión donde se determinará si finalmente la rama candidata se incluye en el proyecto o no.

- Una Pull Request permite a otras personas revisar los cambios que has hecho en una rama de un repositorio git.
- Una vez se abre una Pull Request, es posible debatir y ver qué cambios se han hecho en un proyecto. Se pueden añadir comentarios, sugerencias y en definitiva, que esté todo el mundo de acuerdo en que ese código debe ser aprobado o no.
- Antes de pedir que nos revisen una Pull Request es importante estar seguro de que está lista para ser revisada y también de que cumple con ciertos criterios, como que está bien documentada, que he hecho las pruebas adecuadas. Escribir una descripción completa en una Pull Request ayudará a dar contexto en la revisión.
- Las Pull Requests grandes pueden llegar a estar varios días abiertas y esto es algo que resta mucha agilidad en un equipo. Cuanto más pequeñas sean las Pull Requests, mejor

Una manera de trabajar de forma colaborativa en GitHub a través de pull request sería la siguiente:

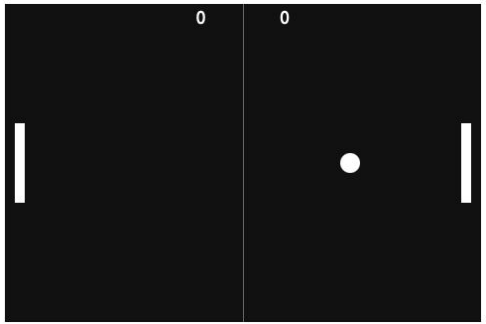
1. **Hacer un fork** del repositorio oficial a tu cuenta de GitHub
2. **Clonar** en el PC en el que vayas a trabajar (local) el repositorio fork. Se crea automáticamente un remote origin apuntando al repositorio fork. **Crear un remote más llamado upstream** que apunta al repositorio oficial (o base), para así poder descargar las actualizaciones del resto de colaboradores (y poder actualizar nuestro fork haciendo push después).

   De esta manera tenemos dos remote, origin que apunta al repositorio de mi cuenta GitHub (el fork que he hecho) y upstream que apunta al repositorio original (al que hice fork).

3. **Se crea una nueva rama** y trabajaremos en las nuevas características, haciendo los commits que hagan falta.

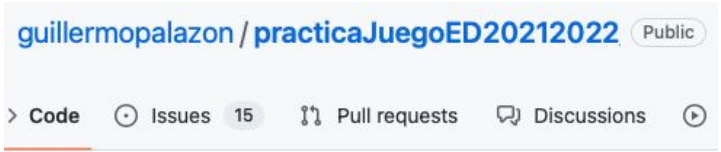
4. Hacer **push de la rama creada** con los cambios al repositorio fork una vez finalizados. Puesto que el repositorio fork es nuestro no tenemos ningún problema de derechos de escritura.
5. En GitHub, crear un **pull request de la rama creada al repositorio oficial o base (upstream).**
6. El administrador del repositorio base upstream, **admitirá** o comenzará una discusión para aceptar los **cambios enviados.**
7. Si hay **conflicto** con los cambios enviados, será necesario bajarse la última versión del repositorio oficial upstream en local (**git pull upstream master (main)**), y luego hacer una merge de la rama master en la rama creada con los cambios: git ckeckout rama - git merge master (main)
8. Dar una **solución a los conflictos existentes**
9. Volvemos a subir la rama con los cambios al repositorio fork, lo cual, actualizará el pull request de manera automática (Github también realiza la gestión de notificaciones a los colaboradores para saber cuando se producen novedades y cambios en el repositorio)
10. Finalmente el administrador del repositorio oficial **acepta los cambios y pasan a formar parte de la rama master oficial**

- Vamos a trabajar en el concepto de Pull Request con una pequeña práctica
- Al clonar el proyecto, si revisamos el contenido de la carpeta descargada podemos ver que tenemos en ella algunos juegos básicos clásicos como el snake (serpiente), un pong o un arkanoid



- Estos juegos no están completos o cada uno de nosotros debemos hacer un pequeño cambio en ellos (que puede ser visual o código).
- Para saber qué tenemos que modificar cada uno de nosotros vamos a hacer uso de las issues de GitHub.
- Un Issue es una nota en un repositorio que nos indica sobre una cuestión a corregir. Puede ser un error que esté ocurriendo, una petición para añadir una nueva opción o característica, una pregunta para aclarar algún tema que no está correctamente aclarado o muchas otras cosas diferentes.
- Para este proyecto he creado una serie de Issue que se corresponden con los cambios/mejoras que cada uno de nosotros debemos abordar respecto al proyecto.
- Para saber el cambio que debemos realizar cada uno, el azar ha asignado NRE con issue, por lo que debemos consultar el listado que existe en el aula virtual (adjunto a la tarea a entregar) y llevar a cabo únicamente la issue que nos ha tocado.

- Ejemplos de issues que nos vamos a encontrar en el repositorio.
- En función del issue que me toque y/o le toque a nuestro compañero es posible que se produzca o no conflicto y tengamos que resolver el mismo.



- Si no lo has hecho (que deberías para haber realizado las tareas anteriores), clona el repositorio https://github.com/guillermopalazon/practicaJuegoED20212022.git
- Debes haber aceptado a la invitación para colaborar en este repositorio
- Ve al repositorio (en un navegador) y elige el issue (tarea) a realizar que te ha sido asignada. Recuerda que debes consultar el documento para saber cuál es la tarea que te toca abordar.
- Entra dentro del issue y arriba a la derecha, en Assignees, elígete a ti mismo como responsable de la tarea (Captura de pantalla 07)

Trabajo en tu repositorio local

1. Si no lo has hecho ya, clona el repositorio
2. Crea una nueva rama nueva, que debe tener de nombre tareaxx (xx será el número de tarea o issue que te haya tocado realizar). Captura de pantalla 08
3. Cámbiate a la rama y realiza las modificaciones que debas en los archivos de la carpeta de trabajo.
4. Realiza un commit con un mensaje descriptivo: Realizada la tarea número xx que consistía en (y colocamos el título del issue). Captura de pantalla 09
5. Ahora vamos a enviar la rama a GitHub, pero antes vamos a traernos de nuevo de internet la versión más moderna del proyecto, ya que un compañero tuyo ha podido ser más rápido que tú y ha podido actualiza la rama master del GitHub, de esta manera se fusionarán lo últimos cambios del proyecto con tu copia local (en la siguiente diapositiva se explican los comandos y capturas a realizar).

- Estando en la rama creada, realiza un git pull origin master (o main, dependiendo de la forma en que esté nombrada la rama principal en el repositorio principal). De esta manera nos traeremos las modificaciones que han hecho los compañeros en la rama master. Captura de pantalla 10
- En caso de que se haya generado conflicto por haber modificado la misma línea de código de un compañero que ya ha integrado su rama en la rama master de github en internet, debes editar el fichero en cuestión y quedarte con la versión apropiada (o mezcla de las dos). Para finalizar el conflicto debes hacer git add . y por último un git commit -a -m "..."
- Envía tu rama a GitHub. Debemos tener en cuenta el nombre que le hemos asignado a nuestra rama: git push origin tareaxx Captura de pantalla 11

Trabajo en GitHub

1. En el navegador ve a github y entra en la rama que acabas de subir (debes cambiar de rama). Allí deberán estar tus cambios realizados. Captura de pantalla 12
2. Inicia un pull Request para solicitar que tu rama sea fusionada con la master (Captura de pantalla 13). Un revisor la comprobará y fusionará los cambios de tu rama en la master, en caso de que todo está bien.
3. Una vez que el revisor haya aceptado tu pull Request (en ningún caso debes hacerlo antes) debes ir al ISSUE de la tarea y cambiar su estado a cerrado, indicando un comentario oportuno (Captura de pantalla 14)

Trabajo local

1. Sitúate en la rama master en local (si no te encuentras ya) y borra la rama de tu tarea en local, para ello harás uso de git branch -d tarea xx (Captura de pantalla 15)
2. Debes dejar pasar un tiempo y hacer un git pull para traerte todas las actualizaciones del proyecto y así no te quedes atrás (Captura de pantalla 16)
3. Mostrar el listado de todas las actualizaciones que se han realizado en el repositorio (git log con los parámetros que veas más oportunos) y comentar el contenido del resultado (Captura de pantalla 17). NOTA: Esta rama también se podría eliminar en GitHub (no lo hagas que pueda ser revisada en la corrección).
4. Ejecuta los juegos y muestra los cambios que han sido realizados (tanto por tu parte como por la de tus compañeros). Incorpora las capturas que sean necesarias.