UD1.Sistema de control de versiones y proyectos

Material seleccionado de repositorios externo, fuente: pedro prieto.

Contenidos a tratar:

Índice:

* Breve introducción a Git
* Entornos: consola y escritorio
* Instalación
* Configuración
* Creación de repositorios
* Cómo guardar cambios
* Historial de cambios
* Cómo deshacer cambios
* Etiquetado
* Ramas
* Fusiones de ramas
* Conflictos
* Remotos
* Flujos de trabajo con ramas

**Instalación**

* <https://git-scm.com/download>

[**Referencias**](https://github.com/pedroprieto/curso-github/blob/gh-pages/sesion-1.org#referencias)

* [Libro de Git](https://git-scm.com/book/es/v2/)
* [Hoja de referencia de Git](https://training.github.com/)
* [Hoja de referencia de Git (PDF)](https://training.github.com/downloads/es_ES/github-git-cheat-sheet.pdf)

**Configuración**

# Opciones obligatorias (nombre y correo)

git config --global user.name "Nombre y apellido"

git config --global user.email CORREO@ELECTRONICO

**Creación de repositorios**

Para crear un repositorio hay que situarse en la carpeta deseada y ejecutar:

git init

**Ciclo de vida**

[](https://camo.githubusercontent.com/2f5084d2a28564283a8dff925fdf8fcf1e33377c1e879c33f94d2316f324654c/68747470733a2f2f6769742d73636d2e636f6d2f626f6f6b2f656e2f76322f696d616765732f6c6966656379636c652e706e67)

**Revisando el estado**

git status

Esquema de colores:

* **Rojo** - Identifica los archivos **modificados o nuevos**. Si se crean archivos dentro de carpetas nuevas, git status solo mostrará el nombre de la carpeta, no su contenido. Si se desea ver el contenido de las carpetas nuevas se deberá ejecutar git status -u.
* **Verde** - Identifica los archivos en el **área de preparación**.

**Visualizar cambios**

git diff

git diff <archivo\_o\_ruta>

Este es uno de los comandos más utilizados en git. Nos permite ver los cambios en los archivos del repositorio o en una ruta específica.

**Añadir archivos al área de preparación (stage)**

git add <archivo> # Añadir archivos individuales

git add . # Añadir todos los archivos nuevos o modificados

El **área de preparación** contiene los **cambios que se añadirán a la nueva versión** cuando ejecutemos un commit. Es posible la siguiente situación:

* Modificar un fichero (aparecerá en color rojo al hacer un git status)
* Añadir el fichero al área de preparación mediante git add FICHERO
* El fichero aparecerá en color **verde** al hacer un git status
* Volver a modificar el fichero
* El fichero aparecerá **dos veces** al hacer un git status:
  + En color **verde**, indicando que se ha añadido el **primer cambio** al área de preparación
  + En color **rojo**, indicando que hay un **segundo cambio** posterior que **no se ha incluido** en el área de preparación
* Si se ejecuta un git commit en este momento **solamente se incorporará el primer cambio** al repositorio como nueva versión. El segundo cambio seguirá existiendo (el archivo no habrá cambiado), pero no estará guardado en el commit
* Si se desea agregar el segundo cambio se deberá ejecutar nuevamente git add para añadirlo al área de preparación

**Visualizar cambios de los archivos en el área de preparación**

git diff --staged

git diff --staged <archivo>

Este comando muestra los cambios que se han agregado al área de preparación (diferencia entre la última versión guardada en el repositorio y el área de preparación).

**Confirmar cambios (commit)**

git commit -m "MENSAJE"

Un commit equivale a una nueva **versión** en el repositorio. Cada commit tiene un **identificador único**, denominado hash. Los commits están relacionados entre sí mediante una **red de tipo grafo**.

En la siguiente sesión estudiaremos como volver atrás en la historia para acceder a una versión anterior del repositorio si se desea.

**Ignorar archivos**

* Archivo .gitignore
* Plantillas de archivos [.gitignore](https://github.com/github/gitignore).

Las rutas y nombres de archivo que aparezcan en el fichero .gitignore serán ignoradas por git **siempre que no hayan sido añadidas previamente al área de preparación o al repositorio**. Por ejemplo, si añadimos un archivo al área de preparación mediante git add y a continuación lo añadimos al fichero .gitignore, git lo seguirá manteniendo en el área de preparación, por lo que será incluido en el repositorio si ejecutamos un git commit.

De igual manera, si previamente hemos guardado un archivo en el repositorio mediante git commit y a continuación lo incluimos en el fichero .gitignore, git no lo borrará: será necesario borrarlo del sistema de ficheros (a través de la consola o el navegador de archivos) y añadir los cambios (git add y git commit) para que se borre del repositorio. Una vez borrado, si lo volvemos a crear veremos que git sí que lo ignora si está incluido en el fichero .gitignore.

**Historial de cambios**

git log

git log --graph

Este comando muestra el histórico de los commits del repositorio. Se puede navegar en el listado mediante los cursores y la barra espaciadora. Para salir hay que pulsar la tecla q.

**Ver cambios realizados en anteriores commits**

git show <commit>

Este comando nos permite mostrar los cambios que se introdujeron en un determinado commit. En primer lugar se puede ejecutar git log para buscar el hash del commit que nos interese y a continuación ejecutar git show indicando después el hash del commit correspondiente.

Los hash de los commits tienen 40 caracteres, pero no es necesario copiarlos enteros: basta con indicar entre los [8 y 10 primeros caracteres](http://git-scm.com/book/en/v2/Git-Tools-Revision-Selection#Short-SHA-1) para identificar un commit correctamente.

**Quitar archivo del área de preparación**

git reset <archivo>

En ocasiones nos encontramos con que hemos añadido cambios al área de preparación que no queremos incorporar al commit. Para ello podemos utilizar este comando, que elimina los cambios del fichero correspondiente del área de preparación. **Los cambios no se pierden** en ningún caso.

**Eliminar las modificaciones con respecto al último commit**

# ¡PELIGRO! Todos los cambios que se hayan hecho al archivo desde el último commit se eliminarán

git checkout -- <archivo>

Este comando es peligroso, ya que **elimina todos los cambios del archivo** que no hayan sido guardados en el repositorio. Es decir, si el archivo tiene cambios y está en color **rojo**, se perderán dichos cambios. Este comando puede ser útil para dejar un archivo tal como estaba en la última versión guardada del repositorio.

**Etiquetado**

git tag NOMBRE\_TAG

Este comando crea un tag en el commit en que nos encontremos en este momento. Un tag es un **alias** que se utiliza para **hacer referencia a un commit** sin necesidad de saber su hash. Normalmente se utiliza para **indicar números o nombres de versiones** asociadas a un determinado commit. De esta manera podemos **identificar una versión de una manera más amable**.

El nombre de los tag se puede utilizar con los comandos de git: por ejemplo, git show.

**Guardado temporal**

# Guardado temporal de cambios no añadidos al área de preparación

git stash

# Restaurar cambios guardados mediante git stash

git stash pop

En ocasiones se hacen cambios que se desea preservar para más adelante: por ejemplo, trabajamos en una modificación de un fichero y de repente nos avisan de que hay un bug en otro fichero que tiene que ser resuelto inmediatamente. Para no trabajar en ambas tareas a la vez podemos ejecutar git stash: los cambios que tenemos en ese momento y que no están en el área de preparación (es decir, los cambios que están en color rojo) se guardan en un área temporal; al ejecutar git status veremos que no hay ninguna modificación, el directorio de trabajo está limpio.

A continuación trabajamos en el bug, hacemos cambios y al terminar ejecutamos git add y git commit para resolverlo. Una vez resuelto, ejecutamos git stash pop y recuperamos los cambios que estábamos realizando antes de ser interrumpidos: veremos que git status nos muestra en color rojo los archivos que habíamos modificado al principio.

Cuando se pida realizar un *commit* recuerda que previamente hay que añadir los archivos al área de preparación si no se ha indicado antes en las instrucciones. En esos casos, un *commit* significa ejecutar los comandos git add y git commit.

**¡IMPORTANTE!** No utilices el bloc de notas de Windows para editar los archivos de texto de las tareas. Utiliza en su lugar un editor específico. Algunas sugerencias son:

* [Visual Studio Code](https://code.visualstudio.com/)
* [Atom](https://atom.io/)
* [Brackets](http://brackets.io/)
* [Sublime Text](https://www.sublimetext.com/)
* [Notepad ++](https://notepad-plus-plus.org/)

El bloc de notas de Windows utiliza una codificación de caracteres específica de Windows y además incluye la extensión .txt por defecto al final de los nombres de archivo. Por tanto, no es una buena elección para trabajar.

La tarea O1:

1. Instala Git en tu sistema operativo. Adjunta una captura de pantalla en la que aparezca el resultado de la ejecución del comando git --version.
2. Realiza la **configuración de Git** según lo indicado en el tema (nombre, correo electrónico y editor de preferencia). Adjunta una captura de pantalla con el resultado de la ejecución de los comandos de configuración.
3. Crea una carpeta denominada S1R1. Realiza las siguientes acciones en ella:
   1. Crea un repositorio Git.
   2. Crea un fichero denominado libros.txt. Añade tres títulos de libros cada uno en una línea distinta.
   3. Haz un primer *commit*.
   4. Añade dos libros al archivo libros.txt.
   5. Haz un segundo *commit*.
   6. Crea un fichero denominado peliculas.txt. Añade tres títulos de películas a dicho archivo.
   7. Haz una captura de pantalla del comando git status.
   8. Crea un fichero denominado comidas.txt. Añade tres nombres de comidas a dicho archivo.
   9. Haz un tercer *commit* que incluya los archivos peliculas.txt y comidas.txt.
   10. Elimina el archivo comidas.txt desde el navegador de archivos.
   11. Añade dos películas más al archivo peliculas.txt.
   12. Haz una captura de pantalla que muestre los cambios en el directorio de trabajo.
   13. Añade los cambios al área de preparación.
   14. Haz una captura de pantalla del comando git status. Debe indicar que se ha borrado el archivo comidas.txt y que se ha modificado el archivo peliculas.txt.
   15. Haz un cuarto *commit*.
   16. Crea un archivo denominado datos.bak. Añade tres títulos de libros a dicho archivo. **¡IMPORTANTE! No añadas el archivo al área de preparación ni hagas ningún commit.**
   17. Crea una subcarpeta denominada output. Crea un archivo denominado salida.txt en su interior. Escribe tu nombre y apellidos en dicho archivo. **¡IMPORTANTE! No añadas los archivos al área de preparación ni hagas ningún commit.**
   18. Haz una captura de pantalla del comando git status. Deben aparecer el archivo datos.bak y la carpeta output como archivos nuevos (color rojo). Recuerda que, por defecto, git no muestra el contenido de una carpeta desconocida, sino solo el nombre de dicha carpeta; si se desea mostrar los archivos nuevos dentro de carpetas desconocidas se debe ejecutar git status -u.
   19. Crea un archivo .gitignore para que los ficheros con extensión .bak y el contenido de la carpeta output/ no se incluyan en el repositorio.
   20. Haz una nueva captura de pantalla del comando git status. Ahora no deben aparecer los archivos datos.bak y output/salida.txt como archivos nuevos, sino que en su lugar debe aparecer únicamente el archivo .gitignore.
   21. Haz un último *commit* para incluir el archivo .gitignore en el repositorio.
   22. Haz una captura de pantalla que muestre el histórico de cambios del repositorio.

**Contenidos**

**Ramas**

**Definición de ramas**

* Una rama es un **puntero** que apunta a un determinado commit.
* Un repositorio debe tener una rama como mínimo.
* El nombre de la rama que se crea por defecto es master. Este nombre no es especial ni tiene una función o significado especial. En **GitHub** la rama que se crea por defecto si se inicializa un repositorio a través de su interfaz web –se llama main.
* Existe un puntero especial llamado HEAD que apunta a la rama en la que estamos en ese momento.
* Al cambiar de rama se modifica el contenido del directorio de trabajo: éste se muestra tal como estaba en la rama a la que hemos saltado.
* La creación y el cambio de ramas se realizan de forma instantánea: no tienen apenas coste.
* El trabajo con ramas es muy interesante por los siguientes motivos:
  + Se pueden hacer pruebas sin modificar el código en producción.
  + Se puede separar el trabajo en tareas o subproyectos que no afecten unos a otros.
  + Cada miembro del equipo puede trabajar sin ser interferido por los demás.

**Crear ramas**

git branch <nombre\_rama>

Este comando creará una rama nueva con el nombre seleccionado. Si no se indica ningún parámetro adicional, la rama creada apuntará HEAD, es decir, el **último commit de la rama en la que nos encontremos**.

Una alternativa es ejecutar git checkout -b <nombre\_rama>. Este comando creará una rama con el nombre indicado y cambiará a dicha rama.

Es posible crear una rama que apunte a un commit o una rama determinada mediante el comando:

git branch <nombre\_rama> <COMMIT\_ >

git branch <nombre\_rama> <NOMBRE\_RAMA\_EXISTENTE>

**Ver ramas disponibles**

git branch

Este comando mostrará las **ramas locales** existentes en el repositorio. Si se desea ver las ramas existentes en el remoto (ver sección siguiente) se puede ejecutar git branch --all.

**Cambiar de rama**

git checkout <nombre\_rama>

git switch <nombre\_rama>

Cualquiera de los dos comandos anteriores cambiará a la rama indicada. Es posible que el comando falle **si hay cambios en el directorio de trabajo** que no estén integrados en ningún commit: si dichos pueden ser sobreescritos por los cambios de la rama a la que se desea cambiar, git abortará el cambio de rama y nos indicará el problema. En ese caso deberemos crear un commit con los cambios que estemos realizando (o bien guardarlos mediante git stash) y a continuación volver a ejecutar el comando de cambio de rama.

**Fusionar una rama**

Fusionar una rama, en inglés *merge*, consiste en incorporar los cambios presentes en una rama a la rama en la que nos encontramos actualmente. Para realizar una fusión hay que realizar las siguientes acciones:

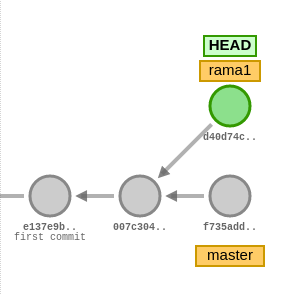
1. Primero nos posicionamos en la rama sobre la que se va a realizar la fusión (la rama que va a recibir los cambios)
2. Para realizar la fusión ejecutar:
3. git merge <nombre\_rama\_a\_fusionar>

Si por ejemplo queremos integrar en la rama principal (master) los cambios presentes en la rama feature1, cambiaremos a la rama principal (si no estamos ya en ella) mediante git checkout master y a continuación ejecutaremos git merge feature1.

**Conflictos**

**Al fusionar una rama** pueden producirse **conflictos**. Un conflicto se produce cuando **diferentes commits introducen cambios en las mismas líneas de los mismos archivos**. Si por ejemplo estamos en un commit referenciado por dos ramas y realizamos desde ese punto común un commit en cada una de las ramas de manera que afecten a la misma línea del mismo fichero, se producirá un conflicto **al fusionar una rama en la otra**.

Por ejemplo, en esta imagen podría producirse un conflicto si queremos incorporar los cambios de la rama master en la rama rama1 mediante git merge master (siempre que los cambios producidos en los commits afecten a las mismas líneas de los mismos ficheros).

[](https://github.com/pedroprieto/curso-github/blob/gh-pages/imagenes/conflicto.png)

Al producirse un conflicto, git no sabe qué cambios deben prevalecer: los de la rama A, los de la rama B, los dos, ninguno, algo totalmente distinto,… En este caso **es necesaria la intervención humana**. Git modificará los ficheros afectados incluyendo **delimitadores** para indicar los cambios que vienen de una rama y los que vienen de HEAD, es decir, de la rama en la que nos encontremos.

Es importante recalcar que **git no perderá información**: la incluirá toda, junto con los delimitadores para identificar la procedencia de los cambios.

Si se produce un conflicto **git quedará en un estado intermedio**: añadirá al área de preparación (color verde) los archivos que no presenten conflictos e **indicará los archivos en conflicto**, en color rojo, para que el usuario los edite y resuelva los conflictos.

Resolver los conflictos pasa por **editar el archivo**, localizar los **delimitadores** y **dejar el archivo como queremos que quede**. Normalmente esta última acción consistirá en decidir **qué cambios son los que queremos dejar y eliminar los delimitadores**. Al final, **el fichero debe quedar tal como queremos que quede**: en ocasiones una de las versiones será la correcta; en otras, la otra versión; en otras, ninguna; en obras, ambas; en otras, algo totalmente distinto.

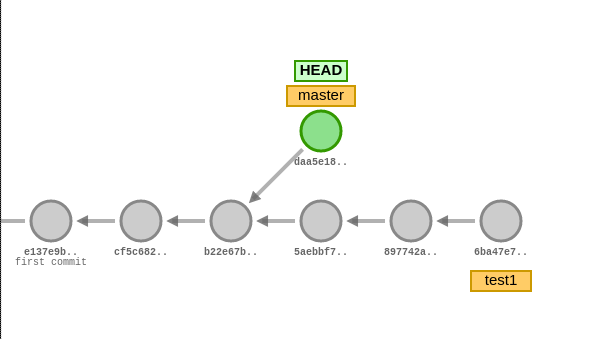
Una vez resuelto el conflicto en todos los archivos habrá que ejecutar los comandos git add y git commit para **crear un commit que resuelva el conflicto**.

**Eliminar una rama**

git branch -d <nombre\_rama>

Este comando eliminará la rama local indicada. **IMPORTANTE**: la eliminación de una rama supone la eliminación del puntero que hace referencia a un determinado commit. Si al eliminar una rama se quedan commits sin referenciar, dichos commits se perderán: pueden recuperarse durante un tiempo solo si se conocen sus *hash*; git realiza también tareas de limpieza al realizar algunas acciones y procede a limpiar los commits “huérfanos”, por lo que pasado un tiempo ni siquiera se podrán recuperar a través de sus *hash*.

En el ejemplo siguiente, si se elimina la rama test1 se perderán los commits 5aebbf7, 897742a y 6ba47e7, ya que no habrá ninguna rama que haga referencia a ellos.

[](https://github.com/pedroprieto/curso-github/blob/gh-pages/imagenes/borrar_rama.png)

**Remotos**

Los repositorios remotos son **versiones** del proyecto que están alojadas en un **servidor web** o en una **carpeta** accesible a través de la **red**.

Los repositorios remotos se almacenan en una forma especial denominada *bare*, que **no dispone de directorio de trabajo**.

Los repositorios remotos pueden estar configurados como **solo lectura** o **lectura/escritura**.

Los repositorios remotos suelen utilizarse como **punto de referencia** de alojamiento del código: es posible realizar **copias** de dicho repositorio remoto denominadas **clones**, que pueden evolucionar de manera **independiente**. Esto hace que git sea un sistema **descentralizado** de gestión de código, ya que **cada clon** es un repositorio **completo** e **independiente** del resto.

Es posible **sincronizar** los cambios de un repositorio remoto a partir de los cambios de un repositorio local y viceversa.

**Clonar un repositorio**

git clone <URL\_REPOSITORIO>

Este comando permite clonar un repositorio remoto a partir de su URL. También es posible indicar una ruta del sistema de ficheros. Por defecto se mostrará la **rama principal** del repositorio remoto. El resto de ramas no aparecen como ramas locales, aunque pueden obtenerse.

Al clonar un repositorio **se crea una referencia al remoto en el repositorio local** con el nombre origin que hace referencia a la URL indicada.

**Ver remotos**

git remote -v

Este comando permite ver los remotos configurados. Otra alternativa es ejecutar git remote show <nombre\_del\_remoto para ver los detalles del repositorio remoto.

**Añadir, eliminar y renombrar remotos**

git remote add <NOMBRE\_REMOTO> <URL\_REPOSITORIO> # Añadir remoto

git remote rm <NOMBRE\_REMOTO> # Eliminar remoto

git remote rename <NOMBRE\_ORIGINAL> <NOMBRE\_NUEVO> # Renombrar remoto

Estos comandos permiten añadir y editar repositorios remotos. Un repositorio local puede tener 0 o más remotos.

**Traer información del remoto**

# Este comando NO realiza la fusión en la rama local

# Si se desean incorporar los cambios habría que realizar un git merge

git fetch <NOMBRE\_REMOTO>

# El nombre del remoto por defecto es ORIGIN

Este comando permite actualizar la información de los repositorios remotos. git se conecta al remoto y comprueba si hay nuevos cambios en las ramas remotas; si es así, actualiza los punteros de las ramas remotas del repositorio local para reflejar dichos cambios. **Pero no incorpora los cambios a las ramas locales**.

Cuando un repositorio tiene configurado un remoto, git crea **en el repositorio local** unos punteros que hacen referencia a las **ramas remotas**. Dichos punteros son similares a las ramas locales, con la peculiaridad de que **no se puede cambiar a una rama remota**. Es decir, las ramas remotas son de **solo lectura**: hacen referencia a la información que tiene git de la situación del repositorio remoto (dónde están las ramas en el repositorio remoto).

En este punto es necesario insistir en la idea de que git es un sistema **descentralizado**: **git no se está ejecutando continuamente** (no es un demonio o un servicio), por lo que no está en contacto continuo con el remoto; si se producen cambios en el remoto (porque otra persona ha realizado un commit en una rama, por ejemplo), **el repositorio local no se sincronizará automáticamente**, sino que para él la rama remota seguirá en el mismo lugar que estaba cuando realizó el último fetch; si a continuación de dicho cambio se ejecuta un nuevo fetch, entonces el repositorio local tomará conciencia del nuevo cambio en la rama remota y **actualizará el puntero de dicha rama remota** al nuevo commit.

En resumen: git fetch es la única manera de que el repositorio local tenga conciencia de que ha habido cambios en el repositorio remoto.

**Traer y fusionar cambios del remoto**

git pull <NOMBRE\_REMOTO> <NOMBRE\_RAMA>

# Pull = fetch + merge

Un pull equivale a dos acciones: un fetch y un merge. Mediante fetch, git comprueba los cambios que hay en las ramas remotas y los refleja en los punteros de las ramas remotas en el repositorio local; a continuación, la acción merge incorpora los cambios de la rama remota a la rama local correspondiente.

Esta operación es una de las más habituales a la hora de sincronizar los cambios con el repositorio remoto. En la mayoría de los casos, las ramas local y remota **están asociadas**: es decir, git ha sido informado de que dicha rama local tiene que sincronizarse con la rama remota correspondiente. Lo más habitual es que ambas ramas (local y remota) tengan **el mismo nombre**. En ese caso bastará con ejecutar git pull sin parámetros: git asumirá que si estamos en la rama rama1, por ejemplo, tiene que sincronizarse con la rama rama1 remota.

Dicho todo esto, es posible **fusionar el contenido de una rama remota en otra rama distinta** a la asociada. Las ramas remotas pueden ser referenciadas como <nombre\_remoto>/<nombre\_rama\_remota>. Así, la rama origin/rama1 hará referencia a la rama rama1 del repositorio remoto origin. Si estamos posicionados en la rama master local podríamos incorporar los cambios de la rama remota origin/rama1 de dos maneras:

* Ejecutando git merge origin/rama1 - Directamente estaremos incorporando los cambios de la rama remota origin/rama1 en la rama local master. En este caso, la rama local rama1 **no recibiría los nuevos cambios que pudiera tener la rama remota** origin/rama1
* Otra opción sería:
  1. Cambiar a la rama **local** rama1: git checkout rama1
  2. Fusionar los cambios de su rama remota asociada: git pull, o git pull origin rama1 si no estuvieran asociadas. En este punto, la rama local rama1 **recibiría los nuevos cambios que pudiera haber** en la rama remota origin/rama1
  3. Cambiar de nuevo a la rama local master: git checkout master
  4. Por último, fusionar los cambios de la rama local rama1: git merge rama1

No hay una manera mejor que otra: simplemente hay que tener claro qué acciones se están ejecutando y qué cambios es están produciendo.

**Enviar cambios al remoto**

git push <NOMBRE\_REMOTO> <NOMBRE\_RAMA>

Esta operación permite **enviar los commits locales de una rama** a la rama indicada del repositorio remoto. Sería el inverso de pull: mientras que pull trae cambios del remoto, push los envía.

Tal como pasa con git pull, la mayoría de las veces las ramas local y remota están **asociadas**, por lo que si ejecutamos git push sin parámetros se enviarán los cambios de la rama en que nos encontremos a la rama remota asociada (la mayoría de las veces tendrán el mismo nombre).

**Ramas asociadas**

Anteriormente hemos hecho referencia al concepto de **rama asociada**. Una rama asociada es **una rama local** configurada para **hacer referencia a una rama determinada del repositorio remoto**. Si una rama está asociada a una rama remota, al situarnos en dicha rama y ejecutar un git push o un git pull haremos referencia a la rama remota asociada correspondiente sin necesidad de tener que especificarla en dichos comandos.

Podemos **asociar una rama** a una rama remota de **varias maneras**:

* Al ejecutar un git clone automáticamente se crea una rama local principal (normalmente con el nombre master o main) asociada a la rama remota principal (master o main, casi siempre).
* Al realizar un **cambio de rama local a una rama existente en el repositorio remoto** mediante git checkout <rama\_existente\_en\_remoto> se creará una rama local **con el mismo nombre que la rama remota** y **se realizará la asociación de dichas ramas**. Por ejemplo, si en el remoto existe una rama principal master y una segunda rama llamada rama1, al realizar el clonado solamente tendremos disponible en local la rama master; si a continuación ejecutamos git checkout rama1 veremos que se crea la rama local rama1, que apunta al mismo commit que la rama remota correspondiente y que ambas ramas quedan asociadas.
* Si creamos una rama local nueva **que no exista en el repositorio remoto**, podemos posicionarnos en dicha rama y a continuación ejecutar el comando git push -u <nombre\_remoṭo> <nombre\_rama> para **crear una rama remota del mismo nombre**, **sincronizarla** con la rama local y **realizar la asociación** de dichas ramas. La opción clave es la opción -u, equivalente a --set-upstream.

**Tareas**

Realiza las tareas que se indican a continuación. Incluye las **capturas de pantalla** que se pidan en un documento tipo **LibreOffice** o **Word**.

Cuando se pida realizar un *commit* recuerda que previamente hay que añadir los archivos al área de preparación si no se ha indicado antes en las instrucciones. En esos casos, un *commit* significa ejecutar los comandos git add y git commit.

**¡IMPORTANTE!** No utilices el bloc de notas de Windows para editar los archivos de texto de las tareas. Utiliza en su lugar un editor específico. Algunas sugerencias son:

* [Visual Studio Code](https://code.visualstudio.com/)
* [Atom](https://atom.io/)
* [Brackets](http://brackets.io/)
* [Sublime Text](https://www.sublimetext.com/)
* [Notepad ++](https://notepad-plus-plus.org/)

El bloc de notas de Windows utiliza una codificación de caracteres específica de Windows y además incluye la extensión .txt por defecto al final de los nombres de archivo. Por tanto, no es una buena elección para trabajar.

Las tareas a realizar son:

1. Crea una carpeta denominada S2R1. Realiza las siguientes acciones en ella:
   1. Crea un repositorio Git.
   2. Crea un fichero denominado actores.txt. Añade tres nombres de actores cada uno en una línea distinta.
   3. Haz un primer *commit*.
   4. Crea una rama denominada test.
   5. Cambia a la rama test
   6. En la rama test crea un fichero denominado actrices.txt. Añade tres nombres de actrices y realiza un *commit* en dicha rama.
   7. Haz una captura de pantalla del resultado del comando git log --graph --all.
   8. Cambia a la rama master.
   9. Incorpora los cambios de la rama test a la rama master. Haz una captura de pantalla de los comandos que has utilizado y de su resultado.
   10. Crea una segunda rama denominada test2. La rama test2 apunta al mismo *commit* que la rama master en este momento.
   11. En la rama master, añade una actriz al fichero actrices.txt y haz un *commit*.
   12. Cambia a la rama test2
   13. En la rama test2, añade una actriz al fichero actrices.txt y haz otro *commit*.
   14. Haz una captura de pantalla del resultado del comando git log --graph --all. Debe haber dos caminos distintos: uno para la rama master y otro para la rama test2.
   15. Cambia a la rama master
   16. Incorpora los cambios de la rama test2 a la rama master. ¿Se produce un conflicto? De ser así realiza una captura del comando git status.
   17. Resuelve el conflicto incorporando los dos nombres de actrices.
   18. Haz una captura de pantalla del resultado del comando git log --graph --all. Observa que se ha creado un nuevo *commit* que integra los dos caminos anteriores.
2. Crea una carpeta denominada S2R2-remoto. Inicializa un repositorio Git en su interior mediante el comando git init --bare. Esta carpeta se utilizará como repositorio remoto.
3. Clona el repositorio S2R2-remoto en una carpeta denominada S2R2. Adjunta captura de pantalla del resultado del comando de clonado. A continuación realiza las siguientes acciones en el repositorio S2R2:
   1. Crea un archivo denominado directores.txt. Añade el nombre de tres directores de cine.
   2. Haz un *commit*.
   3. Realiza un *push* al repositorio remoto. Adjunta captura de pantalla del resultado.
   4. Crea una rama denominada version1.
   5. Cambia a la rama version1.
   6. En la rama version1 añade el nombre de dos directores de cine más al archivo directores.txt y haz un *commit* de los cambios.
   7. Realiza un *push* de la rama al repositorio remoto de manera que **quede asociada a la rama remota del mismo nombre**. Adjunta captura de pantalla del resultado.
4. Clona el repositorio S2R2-remoto en una segunda carpeta denominada S2R3. Realiza las siguientes acciones sobre ella:
   1. Muestra en la consola el contenido del fichero directores.txt y el resultado del comando git status. Debe mostrar tres directores.
   2. Cambia a la rama version1. Muestra el resultado del comando. Comprueba que se crea una rama local version1 con el contenido de la rama remota origin/version1 y enlazada con ella. Al clonar el repositorio la rama no existía (solo se clona la rama principal, master), pero al cambiar a una rama que existe en el remoto se produce su creación local y enlazado con su correspondiente remota.
   3. Muestra el contenido del fichero directores.txt por la pantalla. Comprueba que se muestran los 5 nombres de directores esperados. Adjunta captura de pantalla.
   4. Cambia a la rama master.
   5. Incorpora los cambios de la rama version1 a la rama master.
   6. Sube la rama master actualizada al servidor. Adjunta captura de pantalla del resultado del comando.
5. Vuelve de nuevo a la carpeta S2R2 y realiza las siguientes acciones:
   1. Obtén los cambios que hay en el repositorio remoto **sin fusionarlos en la rama local**. Adjunta captura de pantalla del resultado del comando utilizado.
   2. Actualiza la rama master local con el contenido de la rama master del repositorio remoto. Adjunta captura de pantalla del resultado del comando utilizado.
   3. Comprueba que aparecen los 5 nombres de directores esperados.