PRÁCTICA GUIADA DE CONTROL DE VERSIONES CON Git

Doc: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones

1. Inicializar una carpeta como un repositorio Git: ***Git init***

Para crear un repositorio en una carpeta existente de archivos, puedes ejecutar el comando git init en esa carpeta. Localiza o crea una carpeta en tu disco duro con algunos archivos ***.java*** dentro y ejecuta:

Git init

Anota el mensaje de salida.
Ahora se puede ver que hay una subcarpeta oculta llamada ".git" en el proyecto.
Este es tu repositorio donde se almacenan todos los cambios del proyecto.
Se usa "git init" para convertir una carpeta existente en un nuevo repositorio Git. Se puede hacer esto en cualquier carpeta en cualquier momento.

2. Agregando y subiendo cambios: ***Git add***

Agrega los contenidos de archivos al ***staging area***.
En Git tienes que agregar previamente los cambios realizados al *staging area* para luego poder hacer el *commit* correspondiente (confirmar los cambios). Si el archivo que estás agregando es nuevo, entonces tienes que ejecutar el comando *git add* para añadirlo inicialmente en tu área de preparación. Si el archivo ya está en "seguimiento" también tienes que ejecutar el mismo comando para agregar las nuevas modificaciones en tu staging area.

Una vez iniciado el proyecto, empieza a agregarle archivos y para ello puedes agregar los archivos de la carpeta donde has creado el repositorio uno a uno con

Git add fichero.java

O agregarlos todos a la vez con

Git add .

Utiliza el comando ***git status*** para ver el estado del repositorio y anota los mensajes que devuelve el comando.

Ahora modifica alguno de los ficheros que has añadido al repositorio, editándolo con Netbeans, y vuelve a ejecutar el comando git status. Anota el mensaje de salida.

Mediante el explorador de Windows, copia un nuevo fichero .java a la carpeta del proyecto sometido al control de versiones de Git.

Ejecuta el comando Git status y anota el mensaje de salida. Qué tratamiento le da Git al nuevo fichero?

3. Confirmando cambios en Git: ***Git commit –m mensaje***

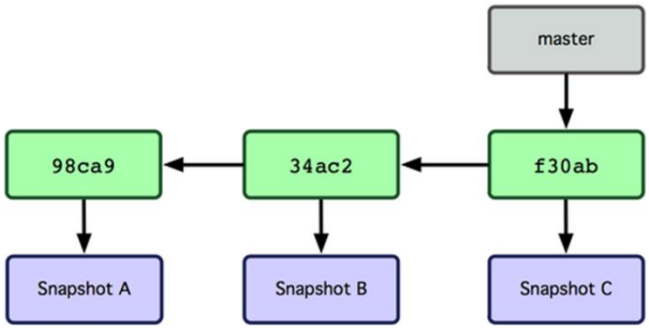
Utiliza el comando git commit para hacer definitivos los cambios de los ficheros que has añadido previamente al área de preparación.
Ejecuta git commit en tu repositorio.
A continuación edita uno de los ficheros que cuyos cambios has confirmado y vuelve a ejecutar git status. Anota el mensaje de salida.

Ahora añade dicho fichero al área de preparación, vuelve a ejecutar git status y anota el mensaje de salida.
Edita el fichero añadido después de git add y antes de confirmar los cambios, consulta el estado del repositorio con git status.
¿Puede estar el mismo fichero preparado y confirmado a la vez? Qué significa esto?

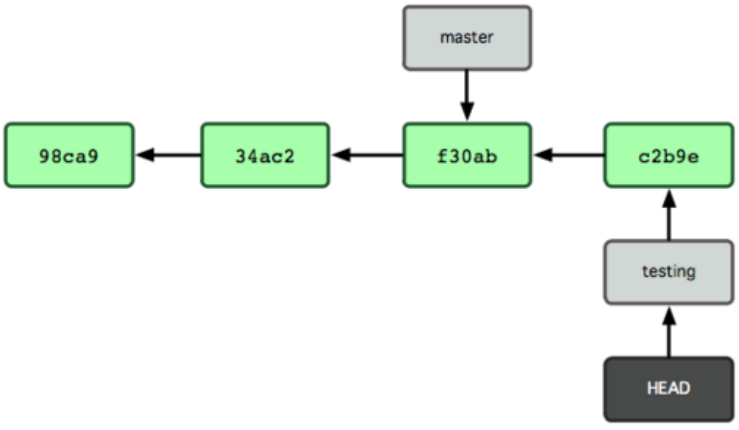
4. Trabajando con ramas en Git: ***Git branch*** , ***Git checkout*** y ***Git merge***.

Uno de los puntos fuertes de Git es su sistema de ramificaciones.
En cada confirmación de cambios (commit), Git almacena un punto de control, un apuntador al nodo correspondiente del árbol de proyecto. Esto permitirá poder regresar posteriormente a dicha instantánea cuando sea necesario.

Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama ***master*** que se creará con la primera confirmación de cambios que realicemos. En cada confirmación de cambios que realizamos, la rama (el puntero que la recorre) irá avanzando automáticamente. Y la rama *master* apuntará siempre a la última confirmación realizada.



Cuando creas una nueva rama se crea un nuevo apuntador para que lo puedas mover libremente. Esta nueva rama o apuntador te permitirá recuperar una línea de trabajo antigua en un momento dado, para retomar el trabajo desde ahí.
Git sabe en qué rama estás en este momento mediante un apuntador especial denominado HEAD.



Puedes incorporar los cambios de una rama a otra, por ejemplo a la rama master, con el comando Git merge:

Git checkout master

Git merge rama1

La fusión de ramas se inicia desde la rama destinataria de dicha fusión de versiones, de ahí que previamente hayamos establecido como rama activa la rama master. Ahora, los cambios realizados están ya en la instantánea (*snapshot*) de la confirmación (*commit*) apuntada por la rama master. Y puedes desplegarlos.

5. Deshaciendo la preparación/confirmación de un archivo: *Git* `reset HEAD` y `Git checkout - -`

Para sacar un archivo del área de preparación ejecutaremos el comando:

git reset HEAD index.html

Para revertir el archivo al mismo estado en el que estaba cuando hiciste tu última confirmación:

git checkout -- index

El uso de este comando es muy peligroso ya que cualquier modificación hecha sobre este archivo ha desaparecido — acabas de sobreescribirlo con otro archivo —. Nunca uses este comando a no ser que estés absolutamente seguro de que no quieres el archivo.

Si lo que queremos es volver a una confirmación anterior a la actual, en la misma rama, podemos hacerlo con las siguientes opciones del comando git reset

a) En este primer caso, queremos desechar los cambios introducidos en esos commits:

   git reset --hard HEAD-3

   Deshace los tres últimos commits de la rama activa.

b) Existe la posibilidad de eliminar uno o varios commits pero manteniendo las modificaciones que contienen dichos commits en el área de trabajo:

   git reset HEAD-1

6. Comprobando diferencias: *Git diff.*

Para ver lo que has modificado pero aún no has preparado escribe:

git diff

Este comando compara tus cambios preparados con tu última confirmación:

git diff —staged

Para instalar una herramienta externa que muestre las diferencias, modificar el fichero ***.gitconfig*** (C:\Users\carmen.CLOUDDOMENI\.gitconfig) del usuario con la ruta donde se encuentra instalada dicha herramienta:

```
[user]
        name = carmen
        email = carmenduran@domenicoscarlatti.es
[diff]
   tool = tkdiff
[difftool "tkdiff"]
   path = "C:/Program Files/Git/tkdiff.exe"
   trustExitCode = false
[merge]
tool = tkdiff
```

7. Configurar un fichero de exclusiones: .gitignore

A menudo tendrás un tipo de archivos que no quieras que Git añada automáticamente o te muestre como no versionado. Suelen ser archivos generados automáticamente, como archivos de log, o archivos generados por tu compilador. Para estos casos es recomendable crear un archivo llamado ***.gitignore***, en el que listas los patrones de nombres que deseas que sean ignorados.
Las reglas para los patrones que pueden ser incluidos en el archivo .gitignore son:

- Las líneas en blanco, o que comienzan por #, son ignoradas.
- Puedes usar patrones *glob* estándar.
- Puedes indicar un directorio añadiendo una barra hacia delante (/) al final.
- Puedes negar un patrón añadiendo una exclamación (!) al principio.

Los patrones *glob* son expresiones regulares simplificadas que pueden ser usadas por las shells. Un asterisco (*) reconoce cero o más caracteres; [abc] reconoce cualquier carácter de los especificados entre corchetes (en este caso, a, b, o c); una interrogación (?) reconoce un único carácter; y caracteres entre corchetes separados por un guión ([0-9]) reconoce cualquier carácter entre ellos (en este caso, de 0 a 9).
Ejemplo de archivo .gitignore:

```
# a comment – esto se ignora
*.a             # no .a ficheros
!lib.a          # pero prepara lib.a, aunque se haya pedido ignorar ficheros.a
*.[oa]          # ignora ficheros con extension .o y .a
*~              # ignora ficheros temporales que terminan en ~
/TODO           # ignora sólo el fichero TODO
build/          # ignora todos los ficheros en el directorio build/
doc/*.txt       # ignora doc/notes.txt, pero no doc/server/arch.txt
```

8. Copiar un repositorio Git: ***Git clone***

Si tienes que colaborar con alguien en un proyecto, o si deseas obtener una copia de un proyecto externo para poder ver o usar el código, debes clonarlo. Para lograrlo sólo tienes que ejecutar el comando *git clone* con la URL del proyecto que deseas copiar o la ruta si está en el mismo equipo.

Para clonar un repositorio en local, basta con copiar la carpeta completa del repositorio en otra ubicación del disco.

También puedes aplicar el comando *Git clone* al repositorio local, indicando la ruta de repositorio que quieres clonar desde la carpeta destino de dicha copia.

git clone ruta_proyecto_a_clonar
git clone url_proyecto_a_clonar

Esto copiará toda la historia de este proyecto, con lo cual lo tendrás a nivel local y te dará una carpeta de trabajo de la rama principal de ese proyecto para que puedas ver el código o empezar a editarlo. Si cambias a la nueva carpeta, puedes ver la subcarpeta .git (donde están todos los datos del proyecto).

9. Trabajar con repositorios remotos Git: ***Git push, Git pull, fork***

Si no quieres realizar todo el trabajo de preparar tu propio servidor Git, tienes varias opciones para alojar tus proyectos Git en una ubicación externa dedicada. Esto suele ser rápido de configurar y sencillo de iniciar proyectos dentro del mismo.
Hay un gran número de opciones de alojamiento, cada una con sus ventajas y desventajas. Para obtener una lista actualizada, puedes mirar en la página Git Hosting del wiki principal de Git:

http://git.or.cz/gitwiki/GitHosting

GitHub es el mayor sitio de alberge público de proyectos Git de código abierto. Y es también uno de los pocos que ofrece asimismo opciones de alojamiento privado en un mismo emplazamiento.

GitHub gira en torno a los usuarios. Esto significa que, cuando alojo mi proyecto Proyecto1 en GitHub, lo encontrarás bajo github.com/carmenduran/Proyecto1. No existe una versión canónica de ningún proyecto, lo que permite a cualquiera de ellos ser movido fácilmente de un usuario a otro en el caso de que el primer autor lo abandone.

Por lo tanto, el primer paso es crear una cuenta de usuario en la plataforma GitHub y añadir tu primer repositorio.

GitHub te permitirá:

a) Iniciar un nuevo proyecto, editando los ficheros en la propia plataforma.
b) Enviar (push) un proyecto Git preexistente en tu equipo:

```
git remote add origin https://github.com/carmenduran/Proyecto1.git
git push origin master
```

c) Importar un proyecto desde un repositorio público Subversion .

Así, el proyecto estará alojado en GitHub. En la página de cabecera de cada uno de los proyectos, podrás ver dos URLs

- HTTPs: Utiliza el protocolo https para enviar (push) /descargar (pull) los proyectos en GitHub
- SSH: Utiliza el protocolo git con pares de claves públicas SSH

Si deseas contribuir a un proyecto ya existente, en el que no tengas permisos de envio (push). GitHub recomienda bifurcar el proyecto. Cuando visitas una página de un proyecto que te parece interesante y con el que deseas trastear un poco, puedes clicar sobre el botón "***fork***" de la cabecera del proyecto, de tal forma que GitHub haga una copia del proyecto a tu cuenta de usuario como colaborador y puedas así enviar (push) cambios sobre él.

EJERCICIOS

1. Crea un nuevo repositorio en Git para prácticar el manejo de ramas.
2. Añade un nuevo fichero de código fuente a dicho repositorio.
3. Edita el fichero y confirma los cambios varias veces. Todos estos cambios se guardan en la rama master que crea por defecto Git.
4. Comprueba el histórico de cambios del repositorio (diferentes puntos de control de confirmaciones en la rama master) con
   Git log
5. Crea una nueva rama sobre la rama master, denominada r***ama1*** para seguir otra línea de trabajo
   Git branch rama1
6. Estable como activa la nueva rama.
   Git checkout rama1

7. Comprueba el estado del repositorio y el histórico de cambios. En este instante, coincide con el estado de la rama master.
8. Edita algunos cambios en el fichero de código fuente del repositorio de rama1 y confírmalos repetidas veces. Comprueba, consultando el histórico de cambios, que el puntero HEAD está avanzando sobre las confirmaciones de los cambios realizados en el fichero en esta línea de trabajo que nos proporciona la rama1.
9. Retoma la rama master y edita el fichero fuente para verificar que has retornado a la versión antigua de dicho fichero.

   Git checkout master

   Esto vuelve el apuntador HEAD a la rama maestra, en concreto a la última confirmación realizada sobre ella y revierte los cambios del directorio de trabajo a la instantánea de ese fichero en dicha confirmación. Esto no significa que se haya perdido los cambios sufridos en el fichero a lo largo de las confirmaciones sobre la rama1, de hecho, ambas líneas de edición se pueden fusionar en una sola si se desea.
10. Desde la rama master vamos a fusionar los cambios realizados en las ramas master y rama1 en una única versión del fichero versionado, guardando el resultado de la fusión en la rama master. A continuación, y puesto que los cambios en rama1 ya están incorporados en master, vamos a eliminar dicha rama1.

    Recupera el directorio de trabajo de la rama master si no lo tuvieras:

    Git checkout master

    Fusiona los cambios de la rama master con las actualizaciones realizadas en la rama1:

    Git merge rama1

    Tras la fusión, la rama master apunta al mismo sitio que la rama1.

    Elimina la rama1. Esto lo puedes hacer con la opción -d del comando git branch:

    Git branch –d rama1

    Para ver la última confirmación de cambios en cada rama, puedes usar el comando

    Git branch –v
11. Realiza un fichero de exclusiones en tu repositorio, para que no se añadan ficheros .class y de configuración al área de preparación del repositorio en cada modificación. Recuerda que estos ficheros se encuentran en los subdirctorios \bin, \build y \.settings del proyecto.
    Comprueba que no se añaden, modificándolos tras una compilación y realizando una nueva confirmación del proyecto.

12. Realiza una copia de tu repositorio local a otra carpeta y anota los mensajes de salida.

13. Ahora realiza una copia de un repositorio alojado en la plataforma GitHub a tu disco duro. Busca algún proyecto que te parezca interesante dentro de la categoría Programming Languages (Open source showcases) de la plataforma GitHub:

    https://github.com/explore

    Localiza la **Git Clone URL** y utilízala para clonar el proyecto.
14. Regístrate en la plataforma GitHub y anota tu nombre de usuario y contraseña.
15. Añade un repositorio a tu cuenta GitHub con el fichero README y LICENSE. Añade tu propio fichero index.html a dicho repositorio desde la propia plataforma.
16. Clona tu repositorio de GitHub al equipo local.

    Git clone https://github.com/carmenduran/proyecto1.git

17. Localiza el repositorio clonado. Edita el fichero index.html del repositorio clonado desde GitHub y confirma los cambios varias veces en local.

18. Sube la nueva versión del repositorio clonado, que incluye los cambios confirmados sobre index.html, a la plataforma GitHub para actualizar el repositorio remoto. (Debes tener permisos de escritura sobre el repositorio)

Git push origin master

19. Añade un nuevo fichero al repositorio clonado en la copia local, p.e., index2.html, y vuelve a subir el repositorio a la plataforma de alojamiento externo. Comprueba nuevamente desde GitHub, accediendo al repositorio allí ubicado, que las actualizaciones realizadas sobre el repositorio local se han trasladado al repositorio remoto.

20. Intenta realizar una bifurcación de tu repositorio de GitHub para seguir otra línea de trabajo sobre él.
    Clic sobre botón ***Fork*** de la cabecera del repositorio.