UD00: Control de versiones Git

3. Fuentes de información

1. Git

1.1. Configuración (`git config`)

Establecer el nombre de usuario:

```
git config --global user.name "Your-Full-Name"
```

Establecer el correo del usuario:

```
git config --global user.email "your-email-address"
```

Activar el coloreado de la salida:

```
git config --global color.ui auto
```

Mostrar el estado original en los conflictos

```
git config --global merge.conflictstyle diff3
```

Mostrar la configuración

```
git config --list
```

1.2. Creación de repositorios

1.2.1. Creación de un repositorio nuevo (`git init`)

Este comando crea una nueva carpeta con el nombre del repositorio, que a su vez contiene otra carpeta oculta llamada .git que contiene la base de datos donde se registran los cambios en el repositorio.

`git init <nombre-repositorio>` crea un repositorio nuevo con el nombre .

1.2.2. Copia de repositorios (`git clone`)

A partir de que se hace la copia, los dos repositorios, el original y la copia, son independientes, es decir, cualquier cambio en uno de ellos no se verá reflejado en el otro.

`git clone <url-repositorio>` crea una copia local del repositorio ubicado en la dirección .

1.2.3. Añadir cambios a un repositorio

Con Git, cualquier cambio que hagamos en un proyecto tiene que pasar por tres estados hasta que guarde definitivamente en el repositorio.

Directorio de trabajo: Es el directorio que contiene una copia de una versión concreta del proyecto en la que se está trabajando. Puede contener ficheros que no pertenecen al repositorio.

Zona temporal de intercambio (staging area): es una zona donde se guardan los cambios temporalmente desde el directorio de trabajo antes de hacerlos definitivos y registrarlos en el repositorio.

Repositorio: Es donde finalmente se guardan los cambios confirmados desde la zona temporal de intercambio.

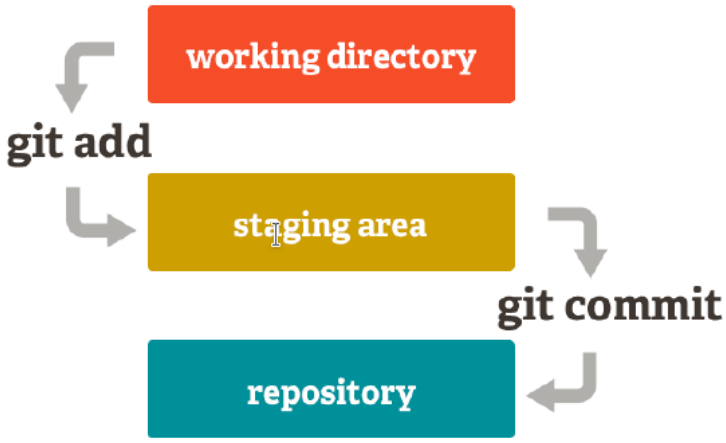
1.2.4. Añadir cambios a la zona de intercambio temporal (`git add`)

`git add <fichero>` añade los cambios en el fichero del directorio de trabajo a la zona de intercambio temporal. `git add <carpeta>` añade los cambios en todos los ficheros de la carpeta del directorio de trabajo a la zona de intercambio temporal. `git add .` añade todos los cambios de todos ficheros no guardados aún en la zona de intercambio temporal.

1.2.5. Añadir cambios al repositorio (`git commit`)

`git commit -m "mensaje"` confirma todos los cambios de la zona de intercambio temporal añadiéndolos al repositorio y creando una nueva versión del proyecto. `"mensaje"` es un breve mensaje describiendo los cambios realizados que se asociará a la nueva versión del proyecto.

`git commit --amend -m "mensaje"` cambia el mensaje del último commit por el nuevo mensaje `"mensaje"`.



1.2.6. Registro de cambios

Para guardar los cambios en un repositorio Git utiliza una estructura de tres niveles:

- **Commit** Contiene información sobre el autor, el momento y el mensaje de los cambios.
- **Árbol (tree)** Cada commit contiene además un árbol donde se registran los nombres y rutas de los ficheros en el repositorio cuando se hizo el commit.
- **Blob (binary file object)** Para cada uno de los ficheros listados en el árbol hay un blob, que contiene una instantánea comprimida del contenido del fichero cuando se hizo el commit.

  Si un fichero del repositorio no ha cambiado en el commit, el árbol apunta al blob del fichero del último commit donde el fichero cambió.

1.2.7. Referenciar un commit

Cada commit tiene asociado un código hash de 40 caracteres hexadecimales que lo identifica de manera única. Hay dos formas de referirse a un commit:

- **Nombre absoluto**: Se utiliza su código hash (basta indicar los 4 o 5 primeros dígitos).

- **Nombre relativo**: Se utiliza la palabra **HEAD** para referirse siempre al último commit. Para referirse al penúltimo commit se utiliza **HEAD~1**, al antepenúltimo **HEAD~2**, etc.

1.3. Estado e historia de un repositorio

1.3.1. Mostrar el estado de un repositorio (`git status`)

`git status` muestra el estado de los cambios en el repositorio desde la última versión guardada. En particular, muestra los cheros con cambios en el directorio de trabajo que no se han añadido a la zona de intercambio temporal y los cheros en la zona de intercambio temporal que no se han añadido al repositorio.

1.3.2. Mostrar el historial de versiones de un repositorio (`git log`)

`git log` muestra el historial de commits de un repositorio ordenado cronológicamente. Para cada commit muestra su código hash, el autor, la fecha, la hora y el mensaje asociado. Este comando es muy versátil y muestra la historia del repositorio en distintos formatos dependiendo de los parámetros que se le den. Los más comunes son:

- `--oneline` muestra cada commit en una línea produciendo una salida más compacta.
- `--graph` muestra la historia en forma de grafo.

1.3.3. Mostrar los datos de un commit (`git show`)

`git show` muestra el usuario, el día, la hora y el mensaje del último commit, así como las diferencias con el anterior.

`git show <commit>` muestra el usuario, el día, la hora y el mensaje del `commit` indicado, así como las diferencias con el anterior.

1.3.4. Mostrar el historial de cambios de un fichero (`git annotate`)

`git annotate` muestra el contenido de un fichero anotando cada línea con información del commit en el que se introdujo. Cada línea de la salida contiene los 8 primeros dígitos del código hash del commit correspondiente al cambio, el autor de los cambio, la fecha, el número de línea del fichero y el contenido de la línea.

1.3.5. Mostrar las diferencias entre versiones (`git diff`)

`git diff` muestra las diferencias entre el directorio de trabajo y la zona de intercambio temporal. `git diff --cached` muestra las diferencias entre la zona de intercambio temporal y el último commit. `git diff HEAD` muestra la diferencia entre el directorio de trabajo y el último commit.

1.4. Deshacer cambios

1.4.1. Eliminar cambios del directorio de trabajo o volver a una versión anterior (`git checkout`)

`git checkout <commit> -- <file>` actualiza el fichero `<file>` a la versión correspondiente al commit `<commit>`.

Suele utilizarse para eliminar los cambios en un chero que no han sido guardados aún en la zona de intercambio temporal, mediante el comando `git checkout HEAD -- <file>`

1.4.2. Eliminar cambios de la zona de intercambio temporal (`git reset`)

`git reset <fichero>` elimina los cambios del chero `<fichero>` de la zona de intercambio temporal, pero preserva los cambios en el directorio de trabajo.

Para eliminar por completo los cambios de un chero que han sido guardados en la zona de intercambio temporal hay que aplicar este comando y después `git checkout HEAD -- <fichero>`.

1.4.3. Eliminar cambios de un commit (`git reset`)

- `git reset --hard <commit>` elimina todos los cambios desde el commit `<commit>` y actualiza el `HEAD` este commit. ¡Ojo! Usar con cuidado este comando pues los cambios posteriores al commit indicado se pierden por completo.

  Suele usarse para eliminar todos los cambios en el directorio de trabajo desde el último commit mediante el comando `git reset --hard HEAD`.

- `git reset <commit>` actualiza el HEAD al commit `<commit>`, es decir, elimina todos los commits posteriores a este commit, pero no elimina los cambios del directorio de trabajo.

1.5. Ramas

Inicialmente cualquier repositorio tiene una única rama llamada master donde se van sucediendo todos los commits de manera lineal.

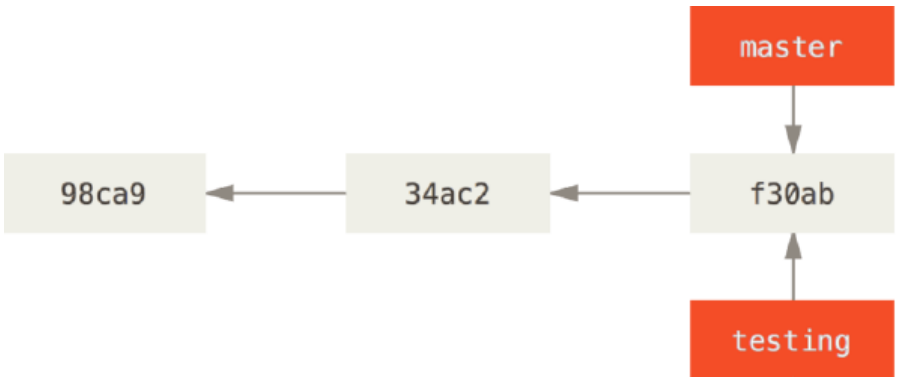
Una de las característica más útiles de Git es que permite la creación de ramas para trabajar en distintas versiones de un proyecto a la vez.

Esto es muy útil si, por ejemplo, se quieren añadir nuevas funcionalidades al proyecto sin que inter eran con lo desarrollado hasta ahora.

Cuando se termina el desarrollo de las nuevas funcionalidades las ramas se pueden fusionar para incorporar lo cambios al proyecto principal.

1.5.1. Creación de ramas (`git branch`)

`git branch <rama>` crea una nueva rama con el nombre `<rama>` en el repositorio a partir del último commit, es decir, donde apunte HEAD. Al crear una rama a partir de un commit, el flujo de commits se bifurca en dos de manera que se pueden desarrollar dos versiones del proyecto en paralelo.



1.5.2. Listado de ramas (`git log`)

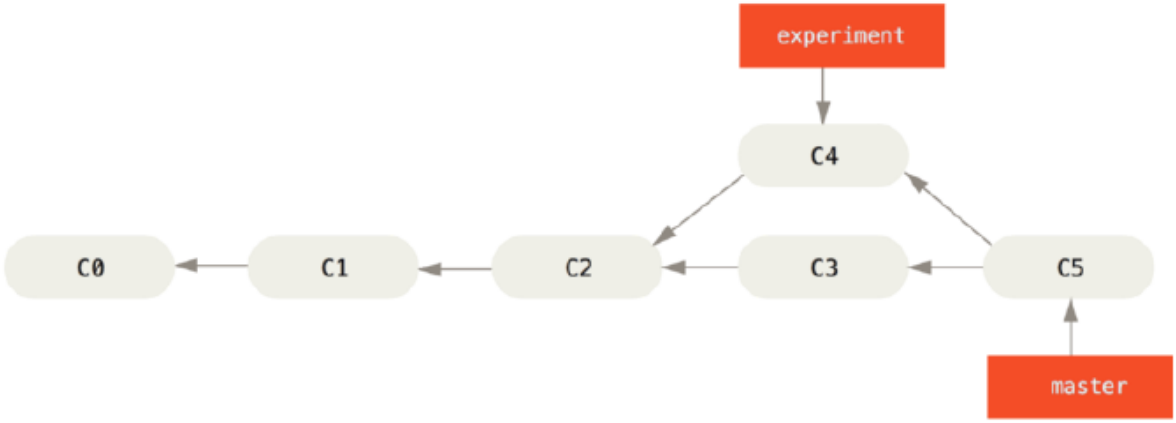
`git branch` muestra las ramas activas de un repositorio indicando con ***** la rama activa en ese momento. `git log --graph --oneline` muestra la historia del repositorio en forma de grafo (`--graph`) incluyendo todas las ramas (`--all`).

1.5.3. Cambio de ramas (`git checkout`)

`git checkout <rama>` actualiza los ficheros del directorio de trabajo a la última versión del repositorio correspondiente a la rama `<rama>`, y la activa, es decir, HEAD pasa a apuntar al último commit de esta rama. `git checkout -b <rama>` crea una nueva rama con el nombre `<rama>` y la activa, es decir, HEAD pasa a apuntar al último commit de esta rama. Este comando es equivalente aplicar los comandos `git branch <rama>` y después `git checkout <rama>`.

1.5.4. Fusión de ramas (`git merge`)

`git merge <rama>` integra los cambios de la rama `<rama>` en la rama actual a la que apunta HEAD.



1.5.5. Resolución de conflictos

Para fusionar dos ramas es necesario que no haya conflictos entre los cambios realizados a las dos versiones del proyecto.

Si en ambas versiones se han hecho cambios sobre la misma parte de un fichero, entonces se produce un conflicto y es necesario resolverlo antes de poder fusionar las ramas.

La resolución debe hacerse manualmente observando los cambios que inter eren y decidiendo cuales deben prevalecer, aunque existen herramientas como `KDif3` o `meld` que facilitan el proceso.

1.5.6. Reorganización de ramas (`git rebase`)

`git rebase <rama-1> <rama-2>` replica los cambios de la rama `<rama-2>` en la rama `<rama-1>` partiendo del ancestro común de ambas ramas. El resultado es el mismo que la fusión de las dos ramas pero la bifurcación de la `<rama-2>` desaparece ya que sus commits pasan a estar en la `<rama-1>`.

1.5.7. Eliminación de ramas (git branch -d)

`git branch -d <rama>` elimina la rama de nombre `<rama>` siempre y cuando haya sido fusionada previamente. `git branch -D <rama>` elimina la rama de nombre `<rama>` incluso si no ha sido fusionada. Si la rama no ha sido fusionada previamente se perderán todos los cambios de esa rama.

1.5.8. Repositorios remotos

La otra característica de Git, que unida a las ramas, facilita la colaboración entre distintos usuarios en un proyecto son los repositorios remotos.

Git permite la creación de una copia del repositorio en un servidor git en internet. La principal ventaja de tener una copia remota del repositorio, a parte de servir como copia de seguridad, es que otros usuarios pueden acceder a ella y hacer también cambios.

Existen muchos proveedores de alojamiento para repositorios Git pero los más usados son GitHub y GitLab.

2. ¿Qué es GitHub?

GitHub es el proveedor de alojamiento en la nube para repositorios gestionados con git más usado y el que actualmente tiene alojados más proyectos de desarrollo de software de código abierto en el mundo.

La principal ventaja de GitHub es que permite albergar un número ilimitado de repositorios tanto públicos como privados, y que además ofrece servicios de registro de errores, solicitud de nuevas funcionalidades, gestión de tareas, wikis o publicación de páginas web, para cada proyecto, incluso con el plan básico que es gratuito.

2.1. Configuración con clave publica/privada

Seguir el manual de github, que resumimos en estos puntos:

2.1.1. Generar la clave (Linux):

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

o:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Después introducir la ubicación del archivo (o dejarlo por defecto) y introducir la `passphrase`.

2.1.2. Añadir la clave a nuestra cuenta github

Siguiendo los pasos de la documentación

2.1.3. Configurar nuestro equipo linux.

Algunas redes bloquean el acceso al puerto `22` (como por ejemplo la red del instituto), por ello `github` tiene habilitada la posibilidad de acceder a `ssh` a través del puerto `443` (típicamente `https`). Para configurar nuestro equipo y acceder a `github` mediante `ssh` con nuestro usuario y nuestra clave debemos configurar el archivo `~/.ssh/config` con el siguiente contenido:

```
Host github.com
Hostname ssh.github.com
Port 443
User martinezpenya
IdentityFile ~/.ssh/pubuntu_github_ssh_key
```

2.2. Añadir un repositorio remoto (`git remote add`)

`git remote add <repositorio-remoto> <url>` crea un enlace con el nombre `<repositorio-remoto>` a un repositorio remoto ubicado en la dirección `<url>`.

Cuando se añade un repositorio remoto a un repositorio, Git seguirá también los cambios del repositorio remoto de manera que se pueden descargar los cambios del repositorio remoto al local y se pueden subir los cambios del repositorio local al remoto.

2.3. Lista de repositorios remotos (`git remote`)

`git remote` muestra un listado con todos los enlaces a repositorios remotos de nidos en un repositorio local. `git remote -v` muestra además las direcciones url para cada repositorio remoto.

2.4. Descargar cambios desde un repositorio remoto (`git pull`)

`git pull <remoto> <rama>` descarga los cambios de la rama `<rama>` del repositorio remoto `<remoto>` y los integra en la última versión del repositorio local, es decir, en el HEAD. `git fetch <remoto>` descarga los cambios del repositorio remoto `<remoto>` pero no los integra en la última versión del repositorio local.

2.5. Subir cambios a un repositorio remoto (`git push`)

`git push <remoto> <rama>` sube al repositorio remoto `<remoto>` los cambios de la rama `<rama>` en el repositorio local.

2.6. Colaboración en repositorios remotos de GitHub

Existen dos formas de colaborar en un repositorio alojado en GitHub:

- Ser colaborador del repositorio:
    1. `Recibir autorización de colaborador` por parte de la persona propietaria del proyecto.
    2. `Clonar` el repositorio en local.
    3. `Hacer cambios` en el repositorio local.
    4. `Subir los cambios` al repositorio remoto. Primero hacer `git pull` para integrar los cambios remotos en el repositorio local y luego `git push` para subir los cambios del repositorio local al remoto.
- Replicar el repositorio y solicitar integración de cambios:
    1. `Replicar` el repositorio remoto en nuestra cuenta de GitHub mediante un `fork`.
    2. `Hacer cambios` en nuestro repositorio remoto.

3. Solicitar a la persona propietaria del repositorio original que `integre nuestros cambios` en su repositorio mediante un `pull request`.

3. Fuentes de información

- Wikipedia
- Code&Coke (Fernando Valdeón)
- Apuntes IES El Grao (Mª Isabel Barquilla?)
- Apuntes IOC (Marcel García)
- Apuntes José Luis Comesaña
- Apuntes IES Luis Vélez de Guevara 17-18 (José Antonio Muñoz Jiménez)
- Introducción a Git (Alfredo Sánchez)