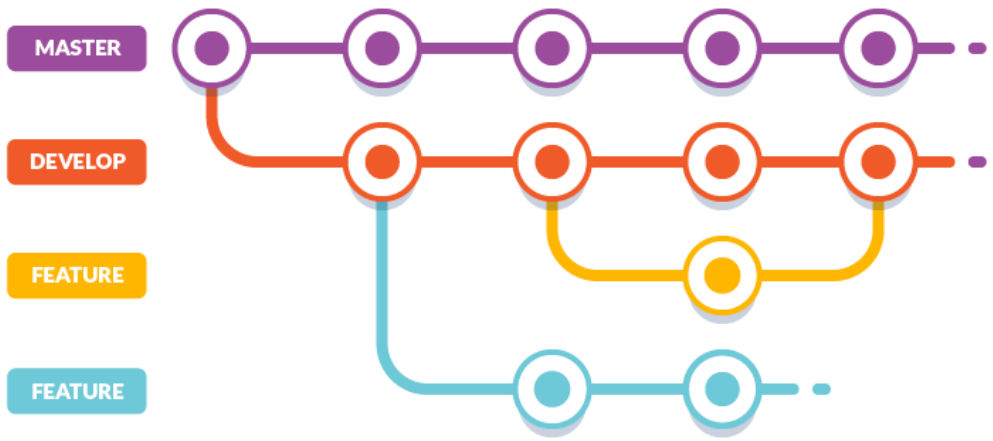
GUÍA GIT BÁSICA

Definiciones

- Git es un software de control de versiones. Su propósito es llevar control de los cambios de un proyecto.
- Con Git, podemos navegar las distintas versiones a través del tiempo, para recuperar una versión anterior si la versión actual tiene errores que no sabemos corregir.
- Git también nos sirve para dividir un proyecto en ramas (*branches*) en las que pueden trabajar individualmente distintos programadores. Cada uno de ellos puede luego integrar los cambios de su rama individual en la rama principal (*master*).
- Normalmente la rama principal no se toca, sino que se crea una rama de desarrollo para, a partir de ahí, crear nuevas ramas para desarrollar nuevas características (*features*). Estas últimas ramas se podrán añadir a la rama de desarrollo, que luego podrá incluirse a la rama principal:



Ejemplo ramas de Git (Fuente: https://www.bitbull.it/en/blog/how-git-flow-works/)

- Las ramas que ya se integraron en la rama principal pueden terminar borrándose, porque ya cumplieron su función. **Es importante no tocar la rama principal (*master*), porque podría ser fatal para el desarrollo, y no hacer cambios directamente en la *master*.** La única excepción a esto es si trabajamos solos en un proyecto, porque en ese caso nosotros hemos llevado individualmente todo el control de las versiones.

Cómo funciona

- El funcionamiento de Git se puede dividir en tres secciones: área de trabajo (local), área de ensayo (*staging area*) y repositorio en línea (GitHub, GitLab…). No podemos enviar los archivos directamente desde el área de trabajo al repositorio

en línea: antes, tenemos que enviarlos al área de ensayo. **Siempre va a haber, por tanto, un paso intermedio entre crear los archivos en un repositorio local y enviarlos al repositorio en línea.**

- Para **crear un repositorio localmente,** utilizamos el comando `git init` en la terminal. Esto crea un área de trabajo en nuestro ordenador. Podemos comprobar que se ha creado el repositorio local porque en la carpeta en la que estemos trabajando se va a crear un archivo oculto llamado `.git`
- Para enviar **todos** los archivos guardados en nuestro repositorio local al área de ensayo, podemos usar los comandos `git add --all`, `git add -A` o `git add .` siempre que nuestra terminal se encuentre abierta en la carpeta donde hemos creado el repositorio local.
- Si solo queremos añadir un archivo, podemos utilizar el comando `git add nombreDelArchivo`
- Si queremos borrar un archivo, podemos utilizar el comando `git rm nombreDelArchivo`
- `git status` nos muestra qué archivos se encuentran en el área de trabajo (qué archivos hemos creado y qué archivos hemos modificado en el repositorio que creamos con el comando `git init`).
- Para enviar los cambios al área de ensayo (*commit*), utilizamos el mensaje `git commit`. **Importante:** Utilizar solo `git commit` hará que la terminal nos devuelva una petición para añadir una descripción de los cambios que hemos realizado en los archivos en este nuevo *commit*. Para ahorrar tiempo, siempre utilizaremos el comando `git commit -m "descripciónCommit"` incluyendo entre comillas ("") un breve mensaje explicitando los cambios que hemos realizado a los archivos.
- Ahora ya podemos subir los cambios que hemos realizado en nuestro repositorio local al repositorio en línea. Utilizamos para esto el comando `git push`

Resumen:

```
git init > it add . > git commit -m "descripciónCommit" > git push
```

Trabajo en equipo

- Al trabajar en equipo, trabajaremos varias personas con el mismo repositorio. Cuando el dueño del repositorio hace `git init` para crearlo, los demás integrantes del equipo no utilizarán `git init` (porque el repositorio ya está inicializado o creado), sino que utilizarán el comando `git clone` para crear una copia (clon) local del repositorio en su ordenador.
- `git branch` crea una nueva rama. Una rama es un nuevo "camino" que adquiere el repositorio. La rama principal es la *master*, pero podemos crear más ramas que no comprometan a la rama principal. En las ramas secundarias que creemos podemos realizar cambios. Si todo sale bien, podemos unir (fusionar) estas ramas secundarias con la rama principal.
- Si surgen problemas en la rama secundaria creada con `git branch`, esta rama se puede borrar sin perjudicar a la rama principal. Para borrar una rama

localmente utilizaremos el comando `git branch -d nombreDeLaRama` siempre que la rama en cuestión haya sido enviada (*pushed*) y fusionada (*merged*) con la rama remota (**Nota:** este *push* y *merge* no se refieren a la fusión de la rama secundaria con la principal, sino con dos versiones de una misma rama secundaria – una que existe localmente y otra que existe en remoto).

- `git checkout` nos permite cambiar de rama. Es importante comprobar que estamos en la rama adecuada para no comprometer la integridad de la rama principal.
- `git merge` es un comando que nos permite unir una rama secundaria con la rama principal.

Repositorio en la nube (en línea): Trabajar con GitHub, GitLab…

- **(!) Importante:** Para proteger la integridad de la rama principal (*master*) en el repositorio en línea, es importante que el dueño del repositorio (el creador del repositorio en GitHub/GitLab) proteja la rama principal. Para hacer esto en GitHub:
    1. El dueño del repositorio tiene que ir al repositorio y hacer clic en `Settings` en la barra superior del repositorio.
    2. Dentro de Settings, en la columna lateral, debajo del título `Code and automation`, hacer clic en `Branches`.
    3. Clicamos en el botón `Add branch protection rule` (“agregar una regla para la protección de ramas”)
    4. En el campo de texto `Branch name pattern`, escribimos `master` (para proteger la rama principal)
    5. Marcamos la primera casilla, `Require a pull request before merging` (“obligar a una petición de envío – *pull* – antes de fusionar las ramas”)
    6. Marcamos, entre las casillas inferiores que se desplegarán, `Require approvals` (“aprobación obligada”: Cada vez que alguien quiera fusionar nuestra rama, al dueño de repositorio le llega un mensaje para que confirme la fusión – *merge* – *una vez*), `Dismiss stale pull requests approvals when new commits are pushed` (siempre que haya *commits* nuevos se borrarán las peticiones de envío – *pull* – que no hayan sido revisadas) y `Require review from Code Owners` (el dueño del repositorio es el único que puede revisar y aprobar el código en las peticiones de envío – *pull*)
    7. Vamos al final de la página y pulsamos el botón `Create`
- Cuando estemos trabajando en una rama secundaria y hayamos acabado la funcionalidad (*feature*) que estemos desarrollando, y queramos que el dueño del repositorio de GitHub acepte nuestros cambios y **fusione** (*merge*) nuestros cambios con la rama principal del repositorio, tendremos que ir a nuestro GitHub. Dentro del repositorio, iremos a la pestaña `Code` (debajo del nombre del repositorio) y haremos clic en `Branches` (en el menú horizontal que aparece encima de los archivos). Dentro de `Branches`, miraremos al menú `Your branches` (“tus ramas”), donde saldrá una lista de las ramas en las que estamos trabajando. En el lateral derecho, clicaremos en el botón que dice `New pull`

`request` para solicitar el envío de nuestros cambios a la rama principal y nos habilite la fusión (*merge*). Nos aparecerá una nueva página donde podremos enviarle un mensaje al dueño de la rama principal. Escribimos el mensaje que queramos (p. ej. "El *login* ya está listo y me funciona correctamente. Por favor, comprueba que funciona. Si no funciona, avísame, si funciona, fusiona la rama") y clicamos en el botón verde `Create pull request`

- Esto hará que se abra una petición para que se revise la rama y, en cuanto haya sido revisada por el dueño del repositorio, este la fusione con la rama principal.
- Para que el **dueño del repositorio** revise y fusione los cambios, entrará en el repositorio y buscará la pestaña `Pull requests` (debajo del nombre del repositorio, dos casillas a la izquierda de `Code`). Dentro de `Pull requests`, verá las ramas que han solicitado cambios. Al clicar en cada rama, saldrán los comentarios y los mensajes de petición de revisión. El administrador tendrá que revisar los cambios clicando en el enlace `Add your review` que aparece a la izquierda de `Review required` (esquina superior derecha del cuadrado). Cuando el código esté revisado, el administrador clicará en `Review changes` (botón verde en la esquina superior derecha de los cuadros de código), añadirá un comentario en el cuadro de texto, marcará `Approve` ("aprobar"), y clicará en el botón verde `Submit review` ("enviar revisión"). También se pueden solicitar cambios marcando `Request changes`.
- Una vez revisados los cambios, el administrador (dueño del repositorio) puede realizar la fusión (*merge*) directamente clicando en `Merge pull request`. Como los cambios han sido aprobados, el colaborador que realizó la petición de revisión también podrá enviar los cambios en la misma pantalla (de la rama que se está modificando) clicando en `Merge pull request` y `Confirm merge` (el botón verde).
- Con esto, la rama que hemos modificado y la rama principal (*master*) estarán fusionadas, y encontraremos todos nuestros cambios en la rama principal. Ahora ya podemos borrar la rama secundaria en la que hemos creado las modificaciones. Así, en la terminal cambiaremos a la rama principal utilizando el comando `git checkout master` (podemos confirmar la rama en la que estamos utilizando el comando `git branch`) y eliminamos la rama secundaria utilizando `git branch -d nombreDeLaRama`. Para comprobar que hemos eliminado la rama secundaria, utilizamos `git branch -a` para ver las ramas existentes en nuestro proyecto. Haciendo solo esto no habremos borrado la rama en el repositorio en línea (GitHub/GitLab) aún, porque nos falta enviar los cambios utilizando el comando `git push origin :nombreDeLaRama` (donde nombreDeLaRama es el nombre de la rama que hemos borrado).

Algunos comandos de Git

- `git config`
  - Se utiliza para configurar el nombre y correo electrónico del autor del repositorio, así como los tipos de archivos usados en el repositorio y otros datos. Por ejemplo:

```
git config --global user.name "nombreUsuario"
git config --global user.email "email@Usuario.com"
```

- `git init`
  - Se utiliza para inicializar (crear) un repositorio y crear el directorio (un archivo oculto) `.git` inicial en un proyecto nuevo o ya existente
- `git clone URL`
  - Se utiliza para crear una copia local de un repositorio Git de una fuente remota a nuestro ordenador. Este es el primer comando que tendremos que usar para crear un clon de un repositorio Git.
- `git add nombreDelArchivo`
  - Añade un archivo específico a nuestra área de ensayo. Para añadir todos los archivos de nuestro proyecto a nuestra área de ensayo, podemos usar los comandos `git add .` , `git add -A` o `git add --all`
- `git commit -m "descripciónCambios"`
  - Envía los cambios realizados en el repositorio local al área de ensayo (*commit*) para poder enviarlos posteriormente al repositorio en línea. Es importante añadir un mensaje que describa los cambios realizados de la manera más corta y específica posible.
- `git status`
  - Muestra los cambios entre el repositorio en línea y lo que hemos enviado con `git commit` al área de trabajo.
- `git remote`
  - Muestra todas las versiones remotas de nuestro repositorio.
- `git checkout nombreDeLaRama`
  - Nos permite cambiar de una rama existente a otra o crear una nueva rama y cambiarnos a ella si utilizamos el comando `git checkout -b nombreDeLaRama`
- `git branch` (https://git-scm.com/docs/git-branch)
  - `git branch -a` muestra una lista de todas las ramas existentes en nuestro proyecto, incluyendo las remotas
  - `git branch nombreDeLaRama` crea una nueva rama
- `git push`
  - Envía todos los cambios al repositorio remoto
- `git pull`
  - Descarga los datos del repositorio en línea en nuestro repositorio local y los fusiona (*merge*) con nuestro directorio
- `git merge nombreDeLaRama`
  - Fusiona (*merge*) una o más ramas específicas en nuestra rama activa y, si no encuentra conflictos, creará un *commit* automáticamente.
- `git diff nombreFuente nombreObjetivo`
  - Muestra los cambios (diferencias) entre nuestra área de trabajo y el repositorio remoto, entre dos ramas, o entre dos archivos locales. Tenemos que añadir siempre el nombre por lo menos un archivo o ítem fuente (A) que se compara con otro archivo o ítem objetivo (B). Por ejemplo:
    `git diff rama1 rama2`
- `git reset`
  - Resetea los directorios locales y remotos al estado del último *commit.*

`git reset --hard` reseteará todos los cambios actuales
`git reset --soft` mantendrá los cambios

- `git revert`
  - Funciona de manera muy similar a `git reset`, pero en vez de resetear los cambios creará un nuevo *commit* que revertirá todo lo que se haya introducido en el *commit* accidental
- `git tag`
  - Se utiliza para marcar un cambio importante en el código, por ejemplo, una versión o un lanzamiento a producción
- `git log`
  - Muestra una lista de los *commits* en una rama específica junto con los detalles correspondientes a cada *commit*

Bibliografía/Referencias

BitBull, "Git Flow: how it works": https://www.bitbull.it/en/blog/how-git-flow-works/
CareerKarma, A Step-By-Step Guide to Git Add: https://careerkarma.com/blog/git-add/
Code War, "Git Curso Completo, Github y trabajo en equipo, todo en 40 minutos": https://www.youtube.com/watch?v=sH9g77J92ns
Dev.to, "Git Commands Ultimate Tutorial [Part 2]": https://dev.to/neshaz/git-commands-ultimate-tutorial-part-2-7ko
FreeCodeCamp, "How to Delete a Git Branch Both Locally and Remotely": https://www.freecodecamp.org/news/how-to-delete-a-git-branch-both-locally-and-remotely/