**ENTORNOS DE DESARROLLO TEMA 2: GIT**

**git config --global user.name “MitoCode”** 🡪 Nombre de usuario.

**git config --global user.email “**[**mitocode21@gmail.com**](mailto:mitocode21@gmail.com)**”** 🡪 Usuario (email).

**git config --global -l** 🡪 Muestra por pantalla los datos anteriores.

**git init** 🡪 Crear un repositorio. Este comando se debe utilizar una vez se esté dentro del directorio que queramos transformar en el repositorio.

**GUARDAR CAMBIOS EN UN REPOSITORIO**

**git status** 🡪 Muestra el estado de los archivos pertenecientes al repositorio.

Si los nombres de los archivos aparecen en rojo, significa que Git no tiene un control de seguimiento actual de estos.

Si los nombres de los archivos aparecen en verde, significa que Git tiene un control de seguimiento actual de estos.

Si no aparecen los nombres de los archivos quiere decir que no se han realizado cambios (NO SE HA HECHO NADA SOBRE LOS ARCHIVOS).

Para conseguir que Git tenga control de seguimiento sobre los archivos (verde), es necesario introducir antes de **git status**, el comando **git add**.

**git add .** 🡪 Guarda los cambios más recientes DE TODOS LOS ARCHIVOS del repositorio para que Git los gestione.

**git add archivo.txt** 🡪 Guarda los cambios más recientes DE UN ARCHIVO del repositorio para que Git los gestione.

**CONFIRMAR CAMBIOS CON UN MENSAJE**

**git commit -m “Primer cambio”** 🡪 Permite escribir un mensaje para informar del cambio realizado.

**git commit** 🡪 Inicia “Virtual Studio Code” con una hoja de commit para que escribamos allí los cambios.

**En resumen:**

**“git add”** para guardar los cambios realizados en un archivo.

**“git status”** para mostrar el estado de los archivos del repositorio.

**“git commit”** para escribir un mensaje que confirme los cambios realizados.

Lo lógico es ejecutarlos de la siguiente manera:

**git add 🡪 git status 🡪 git commit**

**MOSTRAR EL HISTORIAL DE CAMBIOS**

**git log** **🡪** Muestra el historial de cambios realizados en el repositorio.

**HISTORIAL DE CAMBIOS DETALLADO**

**git log --oneline --decorate --all --graph 🡪** Muestra un historial detallado de cambios realizados en el repositorios.

**OTROS COMANDOS IMPORTANTES**

**git diff 🡪** Muestra el último cambio realizado (ANTES DEL “git add”).

En rojo, como estaba el archivo antes del último cambio.

En verde, como está el archivo actualmente.

**git checkout .** **🡪** Revierte el último cambio realizado (ANTES DEL “git add”).

**git commit -- amend -m “Nuevo mensaje” 🡪** Permite cambiar el mensaje del último commit.

**ELIMINAR CAMBIOS**

**git restore --staged archivo.txt** **🡪**  Sacar cambios staging area (DESPUÉS DE “git add”).

**git restore archivo.txt 🡪** Descarta (SE PIERDEN) los cambios en el working directory (DESPUÉS DE 1 COMMIT PERO ANTES DE “git add”).

**RENOMBRAR Y ELIMINAR ARCHIVOS**

**git mv archivo.txt nuevonombre.txt** **🡪** Cambia el nombre de un archivo del repositorio.

**git rm archivo.txt** 🡪 Elimina un archivo del repositorio.

**ARCHIVO .gitignore**

Se crea en “Virtual Studio Code” un fichero llamado **.gitignore.**

El fichero **.gitignore** es un fichero de texto que sirve para poder ignorar ciertos ficheros o directorios que resulten de escasa utilidad o con los que simplemente no queramos trabajar o no queramos modificar.

Dentro del fichero **.gitignore**, si se escribe:

**nombreArchivo** 🡪 Ignora el fichero con ese nombre.

**/nombreDirectorio/** 🡪 Ignora el directorio con ese nombre (y todo su contenido).

.txt 🡪 Ignora todos los ficheros con extensión “.txt”.

El **\*** significa, por así decirlo, “esto, y lo que sea que venga después”.

Ejemplo: tenemos 2 ficheros: **kakatua** y **kakatuaPollo.**

Si en **.gitignore** escribimos: **kakatua\*** se ignorarán tanto el fichero kakatua como kakatuaPollo. Ya que el **\*** significaría en este caso: “kakatua, y lo que sea que venga después”.

**RAMAS Y UNIONES**

Una **rama** es un área contenida en un repositorio que permite una división del estado del código. Es como un área contenida en un repositorio en la que puedes experimentar. La rama por defecto de Git (rama principal) es la rama **“master”**.

Una rama se crea a partir de un archivo ya creado (por ejemplo **archivo.txt**). Por lo que, de forma lógica, se seguiría este esquema para ello:

**Crear archivo 🡪 Nombrarlo como rama (git branch)**

**git branch nombreArchivo** 🡪 Crea una nueva rama en ese archivo.

**git checkout nombreArchivo** 🡪 Permite cambiar a la rama mencionada (DESPUÉS DE “git branch”).

**git checkout -b nombreArchivo** 🡪 Crea y cambia automáticamente a la rama mencionada (Este comando es como un **“git branch” + “git checkout”**).

**git checkout master** 🡪 Permite volver a la rama principal (Estando en una rama diferente, DESPUÉS DE “git branch”).

**git merge nombreArchivo** 🡪 Permite unir la rama mencionada a la rama principal (Todo aquello realizado en la rama mencionada pasa a estar en la principal).

**git branch -d nombreArchivo** 🡪 Borra la rama mencionada (Conviene borrar una rama DESPUÉS de “git merge”).

Cabe aclarar que al borrar una rama, se borra la rama creada a partir de ese archivo creado. SE BORRA LA RAMA, NO EL ARCHIVO. Una vez que la rama se haya borrado ese archivo volverá a ser uno común y corriente.

**Solucionar un conflicto:**

Todos estos comandos funcionan sea la situación que sea, PERO EL COMANDO **“git merge”** PUEDE NO FUNCIONAR SIEMPRE, ya que se puede dar el caso en el que el contenido de la rama secundaria no coincida correctamente con el de la rama principal, lo que alterará el proyecto. A la hora de unir la rama secundaria con la principal, las líneas de código de ambas ENTRARÍAN EN CONFLICTO al no ponerse de acuerdo.

En este caso, la solución sería esta:

IR A LA RAMA PRINCIPAL.

Desde allí, al abrir el archivo de la rama secundaria con “Virtual Studio Code”, en el archivo se encontrarán aquellas cosas escritas desde la rama secundaria y (marcado literalmente con “<<<<<”) aquellas cosas escritas desde la rama principal. Lo único que hay que hacer es retocar el archivo de la forma que queramos para que quede como queramos, es como retocar manualmente las 2 versiones del impuestas en las ramas en un mismo fichero.

Una vez hecho,  **“git add”**, después **“git commit”** con un mensaje tipo “Conflicto resuelto”, y por último **“git merge rama secundaria”**. Tras esto, todo quedaría solucionado.

**ETIQUETAS**

Las etiquetas son referencias que apuntan a puntos concretos en el historial de Git. Generalmente, el etiquetado se usa para capturar un punto en el historial que se utiliza para una publicación de versión marcada (por ejemplo, v1. 0.1).

**Una etiqueta es como una rama que no cambia.**

**git tag nombre\_etiqueta** 🡪 Crea una etiqueta.

**git tag -a nombre\_etiqueta -m** 🡪 Crea una etiqueta acompañada de un commit de referencia (git tag -a v.1.0.0 -m “Versión inicial”).

Al crear una etiqueta en una rama, la rama pasará a tener esa etiqueta al lado de su nombre, indicando que actualmente posee esa versión.

Para saber cómo nombrar un archivo como una versión anterior importante de la rama: [**Curso de Git y Github - 10 Etiquetas (tags)**](https://www.youtube.com/watch?v=s3sbKPlXQz4&list=PLvimn1Ins-43-1sXQmGZPWLjNjPyGNi0R&index=10) **- minuto 3:39.**

**git tag -d nombre\_estiqueta** 🡪 Borra una etiqueta.

**SUBIR ETIQUETAS**

**git push origin nombre\_etiqueta** 🡪 Sube ESA ETIQUETA al repositorio de Github.

**git push --tags** 🡪 Sube TODAS LAS ETIQUETAS al repositorio de Github.

**COMO SUBIR UN PROYECTO LOCAL A GITHUB**

git remote add origin <https://github.com/NOMBRE_USUARIO/NOMBRE_PROYECTO.git>

git branch -M main

git push -u origin main

**COMO SUBIR CAMBIOS A UN REPOSITORIO REMOTO**

git push

**CONFIGURAR VISUAL Y GITHUB PARA MERGE**

git config --global diff.tool vscode

git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"

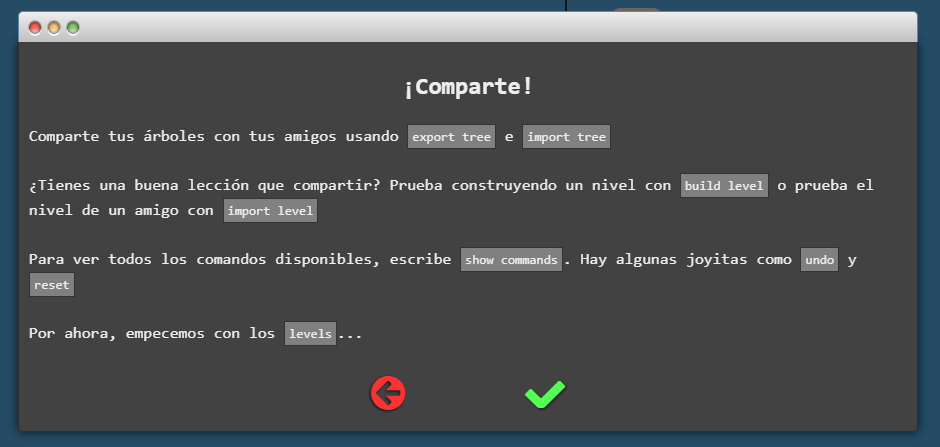
git config --global merge.tool vscode

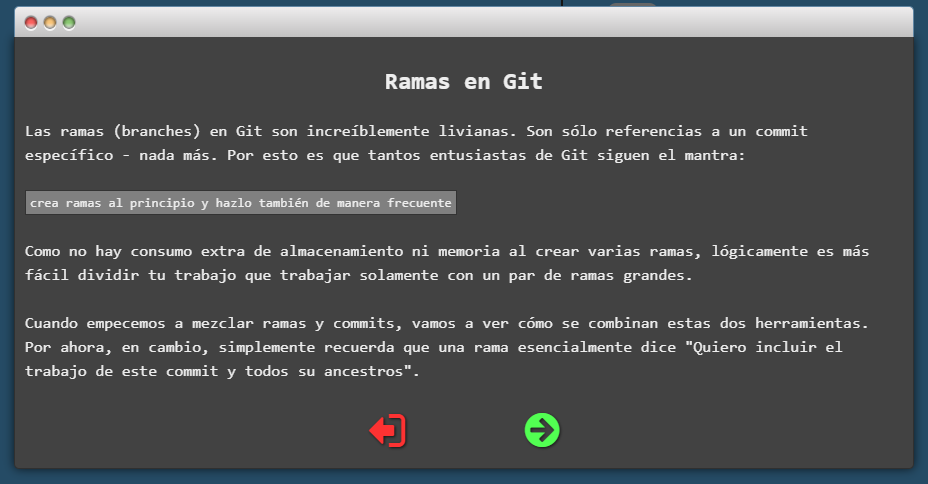
git config --global mergetool.vscode.cmd "code --wait $MERGED"

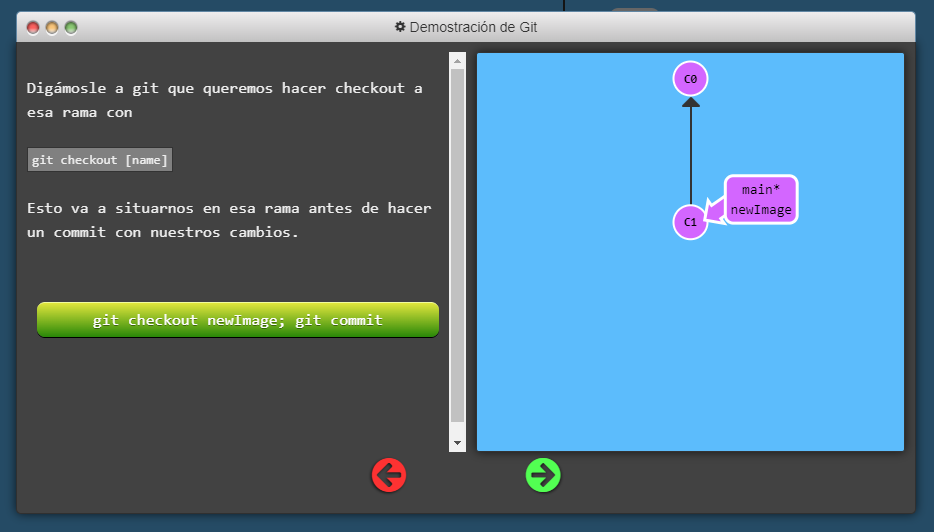
**ENTORNOS DE DESARROLLO TEMA 5: GIT BRANCHING**

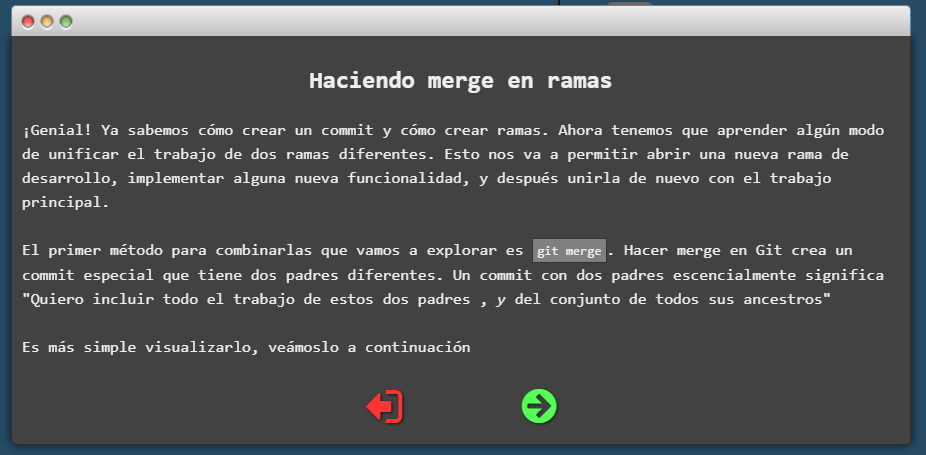
Tienes una gran variedad de comandos de git en este sandbox. Estos incluyen:

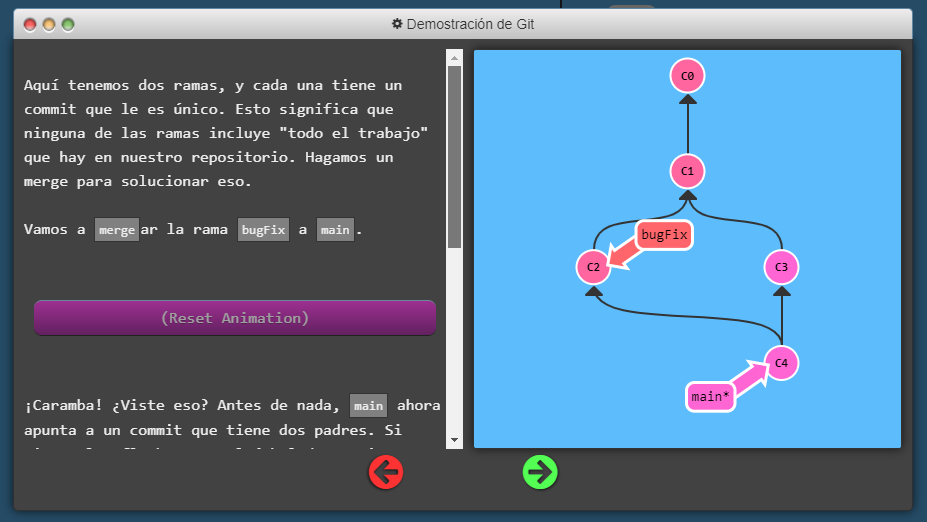
* commit
* branch
* checkout
* cherry-pick
* reset
* revert
* rebase
* merge











git checkout bugFix

git merge main

