

Universidad de El Salvador

Facultad Multidisciplinaria de Occidente – Ingeniería en Desarrollo de Software

| | | |
|--|----------------------|----------------------------|
| Asignatura: Desarrollo de Aplicaciones WEB | Unidad #1 - Semana 4 | Tema: Control de versiones |
| | Fecha: | Autor: |

Introducción

Bienvenidos a la primera lectura de la unidad 1 de Desarrollo de aplicaciones web. En este material, aprenderemos sobre los sistemas de control de versiones (VCS) y los comandos básicos de Git, el VCS más popular. Al final de la lectura, habrá una actividad para poner en práctica lo aprendido.



Contenido

| | |
|--|----|
| 1. Control de versiones..... | 3 |
| 2. Git..... | 3 |
| 2.1. Algunas características clave de Git..... | 3 |
| 2.2. Comandos de Git mas usados..... | 4 |
| 2.2.1. Git clone..... | 4 |
| 2.2.2. Git branch..... | 5 |
| 2.2.2.1. Creando Ramas..... | 5 |
| 2.2.2.2. Mostrar ramas..... | 7 |
| 2.2.2.3. Borrar una rama..... | 7 |
| 2.2.3. Git checkout..... | 8 |
| 2.2.3.1. Navegar entre ramas:..... | 8 |
| 2.2.4. Git status..... | 9 |
| 2.2.5. Git add..... | 11 |
| 2.2.6. Git commit..... | 12 |
| 2.2.7. Git push..... | 13 |
| 2.2.8. Git pull..... | 15 |
| 2.2.9. Git revert..... | 16 |
| Actividad Interactiva del material de lectura..... | 22 |
| Referencias..... | 23 |

1. Control de versiones

Se traduce como "Versión Control System" (VCS). El SCM es una herramienta que ayuda a gestionar los cambios en el código fuente y otros archivos a lo largo del tiempo, permitiendo a los desarrolladores trabajar de manera colaborativa y mantener un historial detallado de todas las modificaciones realizadas en un proyecto de código (Atlassian Corp., S.F.).

2. Git

Es un sistema de control de versiones distribuido ampliamente utilizado para rastrear y gestionar cambios en el código fuente y otros archivos de un proyecto de desarrollo de software. Fue creado por Linus Torvalds en 2005 y se ha convertido en una herramienta esencial para el desarrollo colaborativo y la gestión de proyectos de software. (Atlassian Corp., S.F.)

2.1. Algunas características clave de Git

- Control de Versiones Distribuido
- Historial de Cambios
- Ramificación y Fusión
- Colaboración
- Rastreo de Cambios
- Etiquetas y Versiones
- Compatibilidad con Plataformas
- Rápido y Eficiente

2.2. Comandos de Git mas usados

2.2.1. Git clone

Git clone es un comando para descargar el código fuente existente desde un repositorio remoto (como Github.com, Gitlab.com etc). Realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en tu computador.

Hay un par de formas de descargar el código fuente, pero veremos cómo clonar usando https.

Figura 1

Comando git clone, utilizado para descargar un proyecto de código almacenado en git

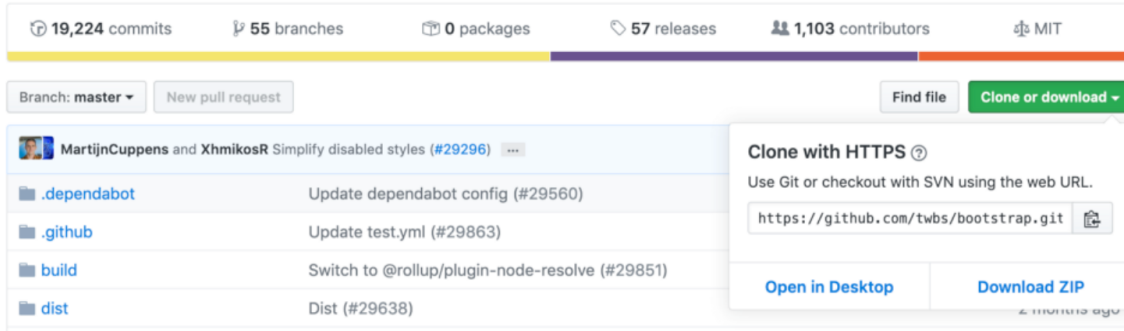


```
git clone <https://link-con-nombre-del-repositorio>
```

Por ejemplo, si queremos descargar un proyecto desde Github, solamente hacemos clic sobre el botón verde (clonar o descargar), copiar la URL del input y pegarla después del comando git clone que he mostrado en la Figura 1, Esto descargara una copia del proyecto en tu espacio de trabajo local y así podrás empezar a trabajar con él.

Figura 2

Captura de pantalla del código fuente en github del framework de frontend bootstrap



Nota. Se muestra una captura del código fuente de Bootstrap en Github, Fuente: github.com

2.2.2. Git branch

Una rama en un sistema de control de versiones (VCS) es una desviación de la línea principal de desarrollo que permite trabajar en cambios sin afectar dicha línea principal. En Git, las ramas son especialmente notables debido a su ligereza, lo que permite crear y cambiar entre ramas de forma casi instantánea. Esto fomenta flujos de trabajo de ramificación y fusión frecuentes, lo que puede transformar la forma en que se desarrolla el software (Atlassian Corp., S.F.).

Las ramas (branch) desempeñan un papel fundamental en el contexto de Git. Permiten que múltiples desarrolladores trabajen simultáneamente en un proyecto. Utilizamos el comando `git branch` para crear, listar y eliminar estas ramas.

2.2.2.1. Creando Ramas

Figura 3

Comando para crear una nueva rama en nuestro proyecto

```
git branch <nombre-de-la-rama>
```

Este comando creará una rama en el repositorio local después de esto estaríamos listos para hacer commit de nuestros cambios en esa rama o para hacer push de nuestro código al repositorio remoto para hacer esto usando la nueva rama al repositorio remoto, necesitaremos usar el siguiente comando:

Figura 4

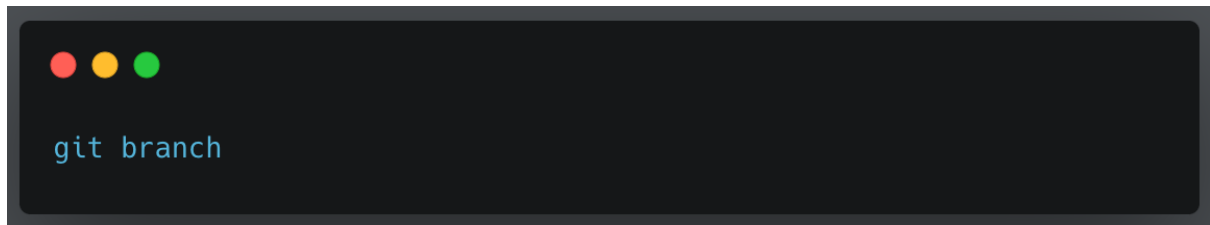
Comando para subir cambios a nuestro repositorio remoto en una rama específica

```
git push <nombre-remoto> <nombre-rama>
```

2.2.2.2. Mostrar ramas

Figura 5

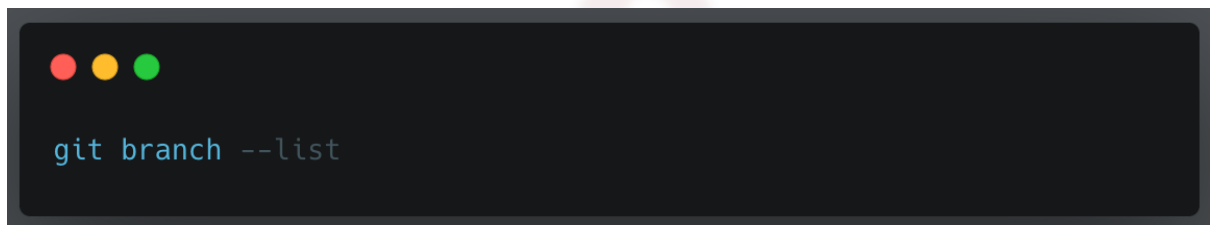
Comando para ver las ramas recientemente creadas



```
git branch
```

Figura 6

Comando Utilizado para ver la lista de ramas creadas en el repositorio

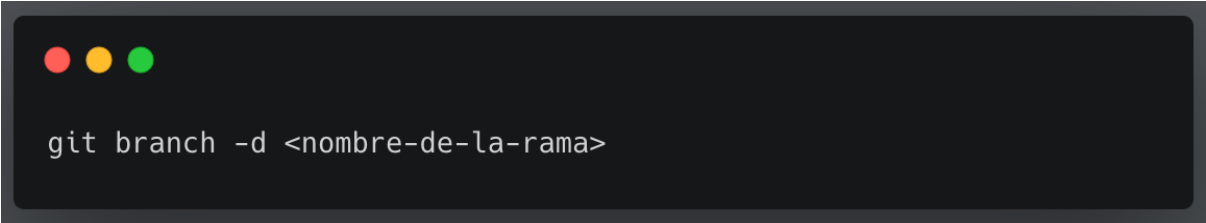


```
git branch --list
```

2.2.2.3. Borrar una rama

Figura 7

Comando para borrar una rama específica a través de su nombre



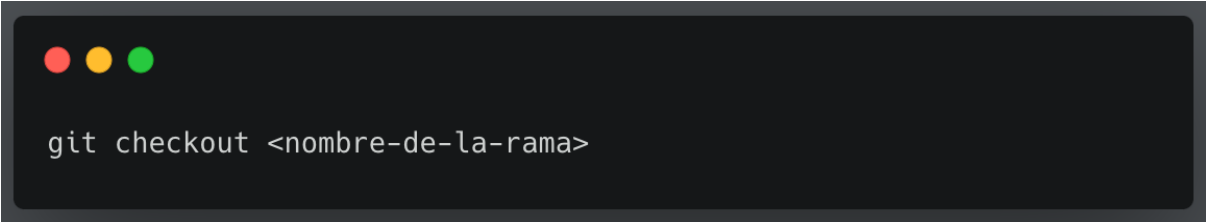
```
git branch -d <nombre-de-la-rama>
```

2.2.3. Git checkout

El comando "Git checkout" es ampliamente empleado en Git y se utiliza principalmente para cambiar de una rama a otra cuando trabajamos en ellas. Además, podemos utilizarlo para examinar archivos y commits.

Figura 8

Comando para cambiar de rama utilizando el nombre de la rama.



```
git checkout <nombre-de-la-rama>
```

2.2.3.1. Navegar entre ramas:

Para cambiar de rama en Git, primero debes confirmar o almacenar los cambios en tu rama actual. Esto se hace con los comandos git commit o git stash. Una vez que tus cambios estén confirmados o almacenados, puedes cambiar a la rama que desees con el comando git checkout <nombre_rama>.

Figura 9

Comando `git checkout -b <nombre_rama>` para crear una nueva rama y cambiarte a ella al mismo tiempo.

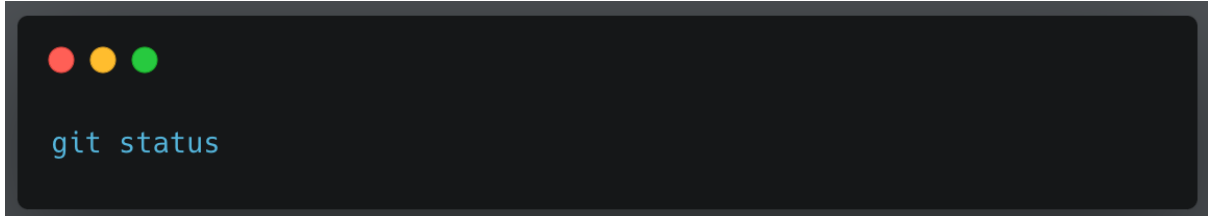


```
git checkout -b <nombre-de-tu-rama>
```

2.2.4. Git status

Figura 10

Comando `status` brinda la información necesaria sobre la rama actual.



```
git status
```

La información mostrada por el comando status, incluye:

- Estado de la rama actual: El comando `git status` muestra el nombre de la rama actual y su estado. Si la rama está actualizada, se mostrará el mensaje "HEAD is up to date". Si la rama no está actualizada, se mostrarán los cambios que necesitan ser confirmados o enviados.

- Cambios que necesitan ser confirmados o enviados: El comando `git status` muestra una lista de los archivos que han sido modificados o eliminados desde la última confirmación. Estos cambios pueden ser confirmados con el comando `git commit` o enviados a un repositorio remoto con el comando `git push`.
- Archivos modificados o eliminados: El comando `git status` muestra una lista de los archivos que han sido modificados o eliminados desde la última confirmación. Estos archivos pueden ser vistos en el modo "unstaged" o "staged". Los archivos en modo "unstaged" aún no han sido confirmados y pueden ser modificados o eliminados nuevamente. Los archivos en modo "staged" están listos para ser confirmados.

Figura 11

Aplicación del comando `git status` el cual nos da información acerca del archivo y las ramas

```
Cem-MacBook-Pro:my-new-app cem$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory
)

        modified:   src/App.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        src/components/
```

Fuente: Ciordia,

<https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/> (2021)

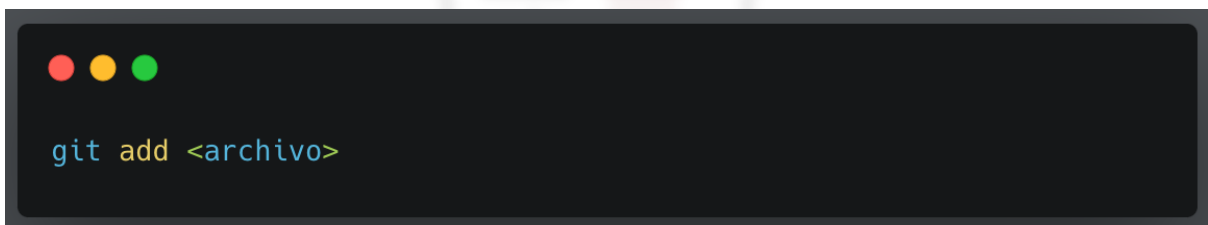
2.2.5. Git add

El comando `git add` agrega cambios de un archivo a la zona de preparación, o índice, de Git. La zona de preparación es una zona temporal donde se almacenan los cambios que aún no han sido confirmados por git para luego pasar al área de staged, cuando creamos, modificamos o eliminamos un archivo, estos cambios se almacenan en el directorio de trabajo de tu repositorio local. Sin embargo, no se incluyen en el próximo commit hasta que se hace uso del comando `git add` para agregarlos a la zona de preparación (Atlassian Corp., S.F.).

Nota: El comando `git add` puede usarse para agregar archivos individuales, directorios o todos los archivos en el directorio de trabajo.

Figura 12

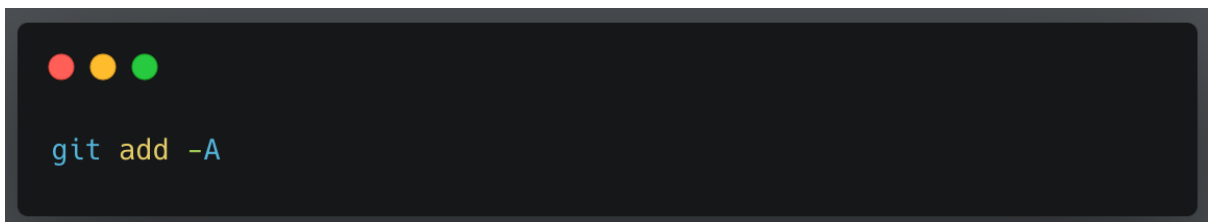
Comando para añadir un solo archivo a la zona de preparación.



```
git add <archivo>
```

Figura 13

Añadir todos los archivos de una vez a la zona de preparación.



```
git add -A
```

Figura 14

Ejemplo de archivos agregados al área o zona de preparación.

```
Cem-MacBook-Pro:my-new-app cem$ git add -A
Cem-MacBook-Pro:my-new-app cem$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   src/App.js
    new file:   src/components/myFirstComponent.js
```

Fuente: Ciordia,

<https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/> (2021)

Tomar en cuenta: git add no cambia el repositorio. Los cambios que no se han confirmado aún se almacenan en el directorio de trabajo local. El comando git add solo prepara los cambios para ser confirmados. Una vez que se ha confirmado un cambio, se almacena en el repositorio y se puede rastrear en el historial de cambios.


2.2.6. Git commit

El comando git commit es una de las herramientas más importantes de Git. Se utiliza para crear una nueva confirmación, que es una instantánea de los cambios que se han realizado en el repositorio. Cada confirmación debe incluir un mensaje que explique los cambios realizados. Este mensaje es importante para rastrear el historial de cambios y comprender lo que se hizo en un momento determinado.

Por lo tanto, es importante usar el comando git commit con frecuencia para capturar los cambios realizados en tu proyecto. Esto te permitirá volver a versiones anteriores del código si es necesario.

Figura 15

Comando para confirmar los cambios del área de preparación.



```
git commit -m "mensaje de confirmación"
```

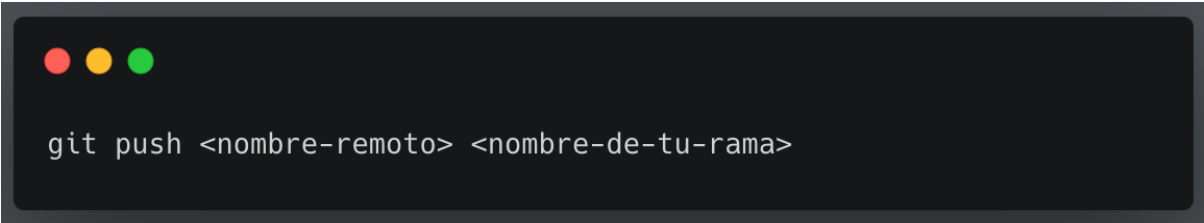
Tomar en cuenta: el comando commit solamente guarda tus cambios en el repositorio local, para publicar tus cambios deberás realizar un push.

2.2.7. Git push

El comando git push se utiliza para enviar los cambios que se han confirmado localmente al repositorio remoto. Esto permite que otros colaboradores del proyecto vean y trabajen con los cambios que has realizado, para usar el comando git push, debemos especificar la rama que deseamos enviar y el nombre del repositorio remoto (Ciordia, 2021). Por ejemplo, para enviar los cambios de la rama main al repositorio remoto llamado origin, usamos el siguiente comando:

Figura 16

Comando para subir los cambios locales al repositorio remoto en una rama específica.



```
git push <nombre-remoto> <nombre-de-tu-rama>
```

Si se ha creado recientemente una rama nueva, es posible que tengamos que empujarla al repositorio remoto antes de poder enviar los cambios de la rama. Esto se debe a que Git no tiene una rama remota que coincida con la rama local de momento. Para empujar una rama nueva al repositorio remoto, podemos usar el siguiente comando:

```
'git push origin [nombre de la rama] --set-upstream'
```

Este comando empujará la rama local al repositorio remoto y establecerá la rama remota como el seguimiento de la rama local, por ejemplo, para empujar una rama nueva llamada new-branch al repositorio remoto llamado origin, usamos el siguiente comando:


```
'git push origin new-branch --set-upstream'
```

El comando git push te pedirá que introduzcas tu nombre de usuario y contraseña para el repositorio remoto. Una vez que hayas introducido tus credenciales, el comando empujará la rama al repositorio remoto y establecerá la rama remota como el seguimiento de la rama local.

Ingeniería en Desarrollo
de Software

Figura 17

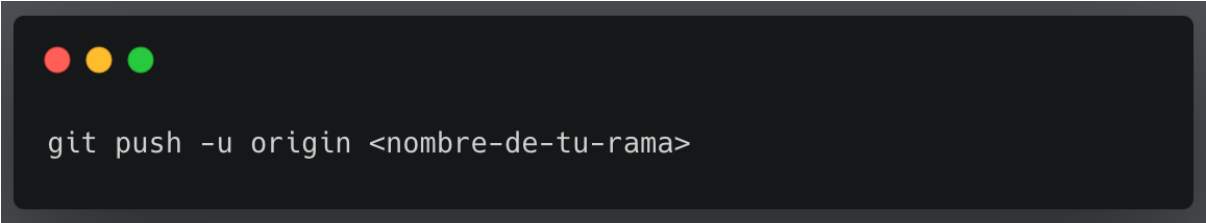
Comando para empujar por primera vez los cambios en el ambiente de trabajo local al repositorio remoto.



```
git push --set-upstream <nombre-remoto> <nombre-de-tu-rama>
```

Figura 18

Comando abreviado para empujar por primera vez los cambios en el ambiente de trabajo local al repositorio remoto.



```
git push -u origin <nombre-de-tu-rama>
```

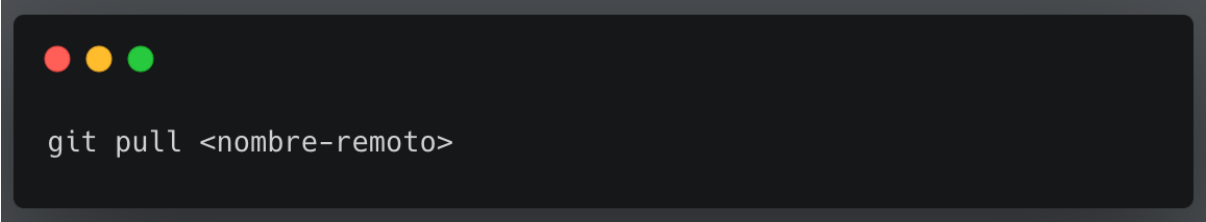
Nota: Git push solamente carga los cambios que han sido confirmados.

2.2.8. Git pull

El comando git pull es una combinación de los comandos git fetch y git merge. git fetch descarga los cambios del repositorio remoto al repositorio local, pero no los fusiona con los cambios locales. git merge permite fusionar los cambios del repositorio remoto con los cambios locales, por lo tanto, git pull es una forma conveniente de actualizar tu repositorio local con los últimos cambios del repositorio remoto (Atlassian Corp., S.F.).

Figura 19

Comando para traer los cambios (commits) desde un repositorio remoto y fusionarlos automáticamente con la rama actual en tu repositorio local



```
git pull <nombre-remoto>
```

Tomar en cuenta: Este comando puede ocasionar que tengamos conflictos en nuestro código que tendremos que resolver manualmente.

2.2.9. Git revert

A veces, es necesario deshacer los cambios que hemos realizado en nuestro código. Hay varias maneras de hacerlo, pero es importante usar los comandos con cuidado para evitar borrar archivos accidentalmente. Una manera segura de deshacer los cambios es usar el comando `git revert`. Este comando crea un nuevo commit que invierte los cambios realizados en el commit especificado.

Para usar el comando `git revert`, primero debemos identificar el commit que deseamos revertir. Podemos usar el comando `git log` para ver el historial de commits. El comando `git log` mostrará una lista de todos los commits, con un resumen de los cambios realizados en cada commit, una vez que hayamos identificado el commit que deseamos revertir, podemos usar el siguiente comando: `git revert [hash del commit]`

Por ejemplo, para revertir el commit con el hash 1234567890, usamos el siguiente comando:

```
'git revert 1234567890'
```

El comando `git revert` creará un nuevo commit que invierte los cambios realizados en el commit especificado. El nuevo commit tendrá el mensaje "Reverted: [nombre del commit]". Es importante tener en cuenta que el comando `git revert` no elimina el commit original. El commit original seguirá estando presente en el historial de cambios.

Algunas recomendaciones para usar el comando: `git revert`

- Siempre haz una copia de seguridad de tu código antes de usar el comando `git revert`.
- Asegúrate de identificar correctamente el commit que deseas revertir.
- Lee el mensaje del commit original para comprender los cambios que se realizarán.

Figura 20

historial de git en la rama master

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
3321844 (HEAD -> master) test
e64e7bb Initial commit from Create React App
```

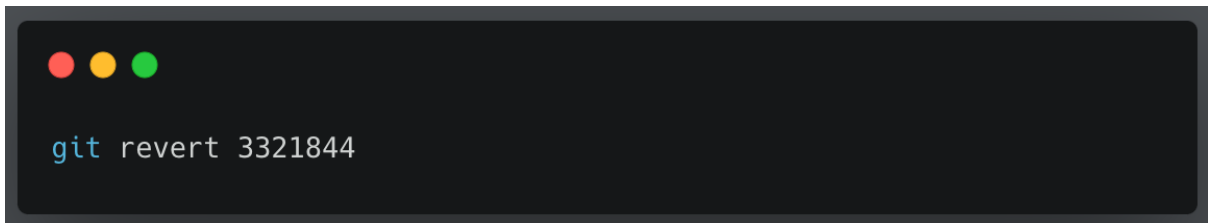
Fuente: Ciordia,

<https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/> (2021)

Luego de esto, solamente necesitamos especificar el código de comprobación que se encontrara junto al commit que queremos deshacer:

Figura 21

Comando revert para revertir los cambios en el id de commit especificado



```
git revert 3321844
```



Después de esto, veremos una pantalla como la siguiente: Para salir de la pantalla, puedes presionar Shift + Q.

Figura 21

Comando revert para revertir los cambios en el id de commit especificado

```
Revert "test"

This reverts commit 332184490ef2b5db289d85ed3f1a13ff2d5f94b9.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Committer: Cem <cem@Cem-MacBook-Pro.local>
#
# On branch master
# Changes to be committed:
#       modified:   src/App.js
#       deleted:    src/components/myFirstComponent.js
#
~
~
~
~
~
~
~
~
~
~/Desktop/my-new-app/.git/COMMIT_EDITMSG" 14L, 384C
```

Fuente: Ciordia,

<https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/> (2021)

El comando `git revert` deshará el commit que le hemos indicado, pero creará un nuevo commit deshaciendo la anterior:

Figura 22

Commit generado con el `git revert`

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
cd7fe6f (HEAD -> master) Revert "test"
3321844 test
e64e7bb Initial commit from Create React App
```

Fuente: Ciordia,

<https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/> (2021)

`Git revert` es una manera segura de deshacer los commits porque no elimina el commit original del historial de cambios. Esto significa que puedes seguir viendo todos los commits, incluso los revertidos. Otra ventaja de `git revert` es que todo sucede en local, esto significa que no podemos revertir accidentalmente commits en el repositorio remoto, por estas razones, `git revert` es la manera preferida para deshacer los commits.

Git merge

Una vez que hayas completado el desarrollo de tu proyecto en una rama, puedes fusionarla con su rama padre. Esto se hace con el comando `git merge` el cual fusiona los cambios de la rama con los cambios de la rama padre. Si no hay conflictos, el comando fusionará los cambios automáticamente. Si hay conflictos, el comando te pedirá que los resuelvas antes de continuar.

Por ejemplo, cuando quieres fusionar tu rama de características en la rama dev:

Figura 23

Comando para movernos a la rama dev



```
git checkout dev
```

Figura 24

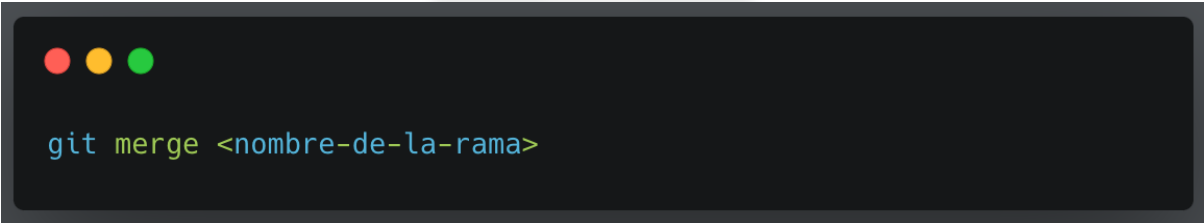
Comando fetch utilizado antes de fusionar las ramas para actualizar la rama dev en el repositorio local.



```
git fetch
```

Figura 25

Comando, para fusionar tu rama de características en la rama dev.



```
git merge <nombre-de-la-rama>
```

Tomar en cuenta: Asegúrate de que tu rama dev está en la última versión antes de fusionar otras ramas, si no, te enfrentarás a conflictos u otros problemas no deseados.

Nota: Para mayor información visitar el siguiente documento de referencia oficial de github: https://training.github.com/downloads/es_ES/github-git-cheat-sheet.pdf

Actividad Interactiva del material de lectura.

Actividad Interactiva

Después de haber leído sobre control de versiones y git y su importancia en la en el desarrollo de software, Crea un nuevo repositorio en tu cuenta de github, descargalo a tu equipo agrega 3 archivos,(index.html, styles.css, app.js)

En index html agrega las etiquetas necesarias para crear un formulario de login.

En styles.css agrega los estilos para decorar tu formulario

En app.js agrega la funcionalidad para dar click a un botón login y que muestre un mensaje en consola, otra etiqueta, un alert o redirigir a otra página html.

Referencias

Atlassian Corp. "Qué es Git." *Atlassian*, S.F,

<https://www.atlassian.com/es/git/tutorials/what-is-git#version-control-with-git>. Consultado el 8 de septiembre de 2023.

Chacon, Scott, and Ben Straub. "1.1 Getting Started - About Version Control." *Git SCM*, S.F.,

<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>. Consultado el 8 de septiembre del 2023.

Ciordia, Nora Gonzalo. "10 Comandos de Git Que Todo Desarrollador Debería Saber."

freeCodeCamp, 11 de Enero del 2021,

<https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/>. Consultado el 9 de septiembre del 2023.

