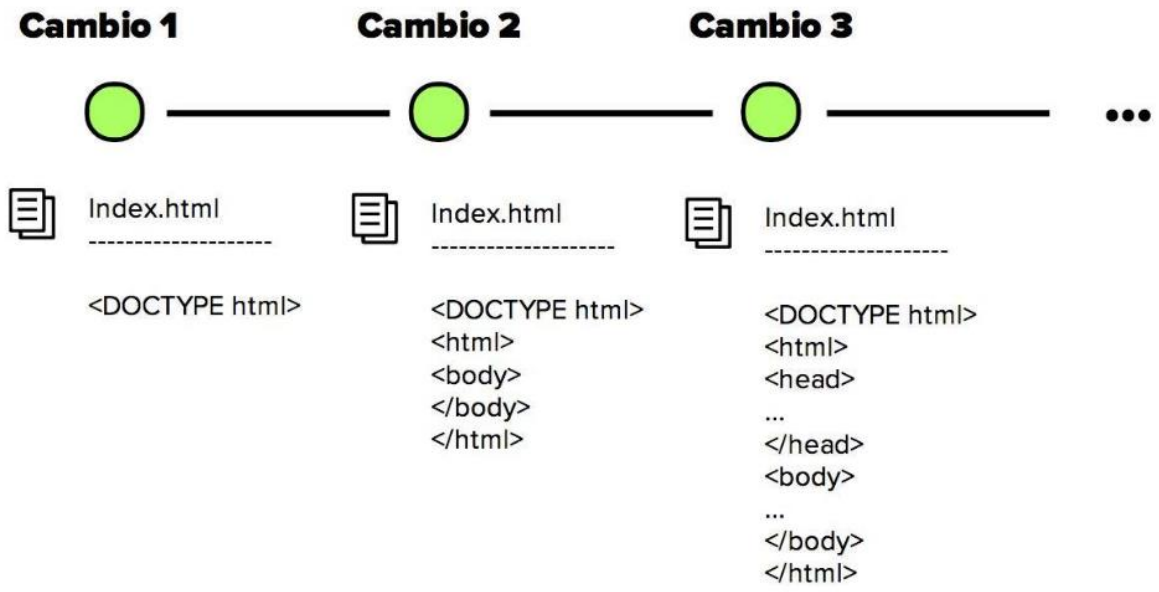
Unidad Temática I. GIT

1. Control de Versiones

1.1 Definición

El control de versiones es un sistema que registra cambios realizados en archivos o conjunto de archivos a lo largo del tiempo, permitiendo que puedas recuperar versiones anteriores de los mismos en caso de ser necesario. Es una herramienta fundamental en el desarrollo de software y en la gestión de proyectos colaborativos, ya que facilita el trabajo en equipo, la coordinación de cambios y la gestión del código fuente.



1.2 Importancia del Control de Versiones

1. **Pilar central de DevOps:** Los sistemas de control de versiones son fundamentales en la metodología DevOps, ya que permiten una colaboración eficiente y rápida iteración del código fuente de un proyecto.

2. **Copias de seguridad constantes:** Trabajar con control de versiones garantiza que siempre haya una copia de seguridad de todo el código y los archivos del proyecto. Cada modificación se registra y se almacena en un historial accesible.

3. **Confirmaciones (commits):** Las actualizaciones de los proyectos se agrupan en confirmaciones, que son registros de los cambios realizados en el código fuente. Esto permite un seguimiento preciso de cada modificación y quién la realizó.

4. **Restauración instantánea:** En caso de errores o regresiones, los usuarios pueden revertir a versiones anteriores de los archivos de manera rápida y sencilla. Esto facilita la corrección de problemas y la restauración de funcionalidades previas.

5. **Registro detallado de cambios:** Los sistemas de control de versiones proporcionan un registro completo de todos los cambios realizados en el proyecto, lo que incluye quién realizó cada modificación y en qué momento.

6. **Colaboración fluida:** Al utilizar un sistema de control de versiones, todos los miembros del equipo pueden trabajar en varios archivos simultáneamente y fusionar fácilmente sus cambios en un repositorio centralizado.

7. **Gestión de errores simplificada:** Con acceso al historial completo del proyecto, es más fácil realizar un seguimiento de los errores y corregirlos de manera efectiva. También facilita la restauración de funciones eliminadas accidentalmente.

1.3 Funcionamiento del Control de Versiones

1. **Rama principal o tronco maestro:** Al crear un repositorio nuevo, se abre la rama principal o tronco maestro. Esta es la rama principal donde la base de código ingresa al canal en el que se compila y se implementa para que lo vea el usuario final.

2. **Ramas:** Las ramas son versiones paralelas de la base de código principal. Bifurcar es el proceso de crear ramas del código a partir de la rama principal. Esto permite que los desarrolladores introduzcan cambios en su propia rama sin afectar directamente la rama principal.

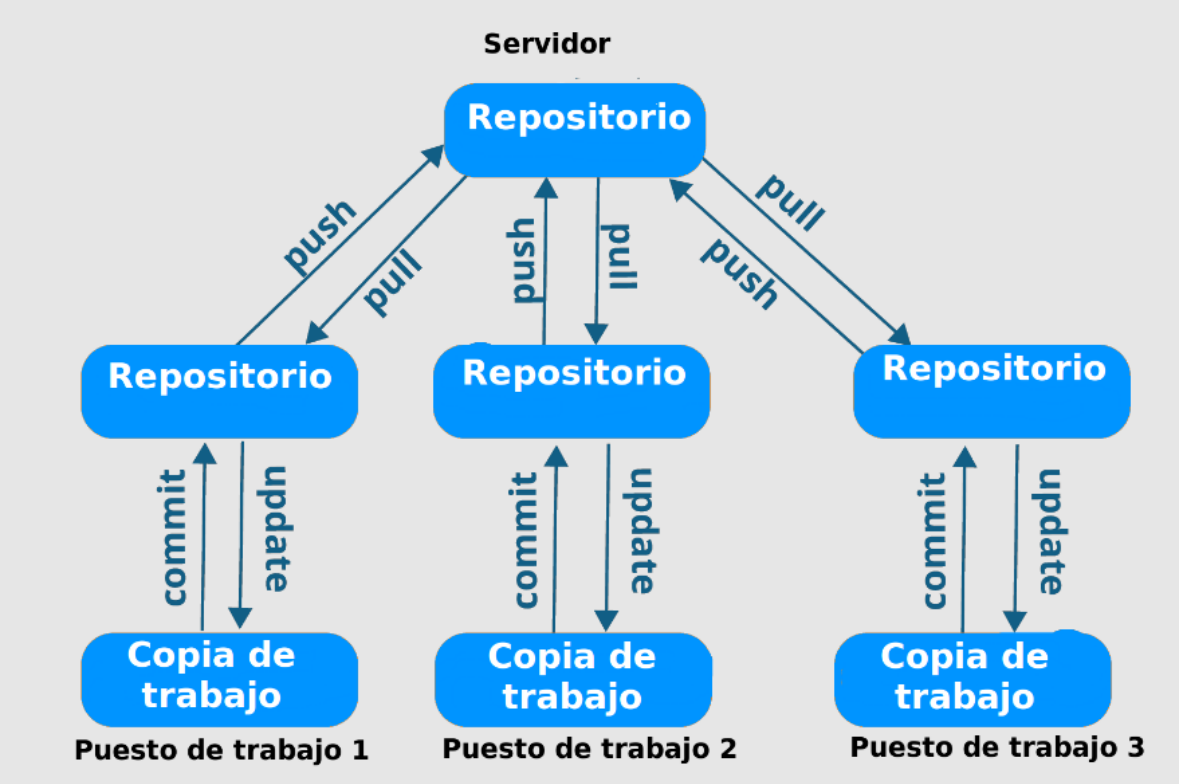
3. **Beneficios de las ramas:** Al crear ramas, los desarrolladores pueden trabajar en cambios específicos sin afectar la rama principal. Esto proporciona un entorno seguro para experimentar y desarrollar nuevas funcionalidades.

4. **Fusionar:** El sistema de control de versiones puede fusionar las ramas separadas en la rama principal una vez que los cambios han sido probados y están listos para ser integrados. Los desarrolladores pueden fusionar sus cambios en la rama principal cuando lo deseen.

5. **Estrategias de bifurcación:** Una estrategia de bifurcación adecuada es fundamental para evitar conflictos en el código y problemas con las compilaciones. Esto implica establecer políticas y procedimientos para la gestión de ramas, la fusión y la resolución de conflictos.

6. **Sistemas de control de versiones:** Los buenos sistemas de control de versiones proporcionan herramientas y funcionalidades que facilitan la

gestión de ramas y la fusión de cambios, lo que permite a los equipos sincronizarse sin problemas y corregir cualquier posible conflicto en el código.



- Pull. La operación de "pull" se refiere a la acción de obtener cambios desde un repositorio remoto y aplicarlos a tu repositorio local. Básicamente, estás "extrayendo" o "recuperando" los últimos cambios que se han realizado en el repositorio remoto y actualizando tu copia local para que esté sincronizada con la versión más reciente del código.

Cuando realizas un "pull", el SCV (como Git) comparará los cambios en el repositorio remoto con los cambios en tu repositorio local y fusionará automáticamente los cambios si es posible hacerlo sin conflictos. Si hay conflictos entre los cambios locales y remotos, el SCV puede solicitar que los resuelvas antes de completar la operación de "pull".

- Push. La operación de "push" se refiere a la acción de enviar tus cambios locales a un repositorio remoto. Básicamente, estás "empujando" tus commits locales al repositorio remoto para que otros miembros del equipo puedan acceder a ellos y colaborar en el proyecto.

Cuando realizas un "push", el SCV enviará tus commits locales al repositorio remoto y los incorporará a la rama específica en la que estás trabajando. Es importante tener en cuenta que, para realizar un "push", necesitas tener los permisos

adecuados en el repositorio remoto. Si no tienes los permisos adecuados, no podrás realizar la operación de "push".

1.4 Tipos de Control de Versiones

1.4.1 Sistemas de Control de Versiones Centralizados.

En este tipo de sistemas, existe un único repositorio central donde se almacena toda la historia y las versiones de los archivos. Los usuarios trabajan con copias locales de los archivos y realizan cambios que luego se sincronizan con el repositorio central.

Ejemplo: Subversion (SVN)

Subversion es un sistema de control de versiones centralizado ampliamente utilizado en proyectos de software. Los desarrolladores trabajan con copias locales de los archivos y utilizan comandos para enviar cambios al repositorio centralizado. SVN gestiona la historia de los archivos y facilita la colaboración en equipo.

1..4.2 Sistemas de Control de Versiones Distribuidos.

En este tipo de sistemas, cada usuario tiene una copia completa del repositorio, lo que significa que no dependen de un servidor central. Esto permite a los usuarios realizar cambios y trabajar de forma independiente, incluso sin conexión a internet. Los cambios se pueden sincronizar entre las diferentes copias del repositorio.

Ejemplo: Git

Git es el sistema de control de versiones distribuido más conocido y utilizado en el mundo del desarrollo de software. Cada desarrollador tiene una copia completa del repositorio en su máquina local y puede realizar cambios, crear ramas (branching), fusionar ramas (merging) y gestionar la historia de los archivos de manera independiente. Git facilita la colaboración en equipos distribuidos y ofrece una potente gestión de versiones.

2. ¿Qué es GIT, GitBash y GitHub?

2.1 Git: Es un sistema de control de versiones distribuido que fue creado por Linus Torvalds en 2005 para el desarrollo del kernel de Linux. Permite a los desarrolladores trabajar en proyectos de software de manera colaborativa, manteniendo un registro de los cambios realizados en los archivos a lo largo del tiempo.

2.1.1 Características.

- **Distribuido**: Cada usuario tiene una copia completa del repositorio, lo que permite trabajar de forma independiente y sin conexión a internet.

- **Ramificación (branching) y fusión (merging) eficientes**: Permite crear ramas para trabajar en nuevas características o arreglos de errores sin afectar la rama principal (por lo general llamada "master" o "main"). Luego, es posible fusionar los cambios de una rama a otra de manera controlada.

- **Velocidad y eficiencia**: Git está diseñado para ser rápido incluso con proyectos de gran tamaño.
- **Gestión de versiones**: Permite llevar un registro detallado de los cambios realizados en los archivos a lo largo del tiempo.
- **Facilita el trabajo colaborativo**: Permite a múltiples desarrolladores trabajar en un mismo proyecto sin interferirse entre sí.
- **Amplia comunidad y soporte**: Git es ampliamente utilizado en la industria del desarrollo de software, por lo que hay una gran cantidad de recursos disponibles y una comunidad activa que puede proporcionar soporte.

2.2 GitBash: Es una interfaz de línea de comandos que proporciona una terminal de Unix para sistemas Windows. Permite a los usuarios ejecutar comandos de Git y trabajar con repositorios Git, ya que viene con su instalación incorporada.

2.3 GitHub: Es una plataforma de desarrollo colaborativo basada en la nube que utiliza Git como sistema de control de versiones. Permite a los desarrolladores alojar y revisar código, gestionar proyectos, colaborar en equipo y realizar un seguimiento de los cambios en el código. GitHub cuenta con una amplia comunidad de desarrolladores y ofrece integraciones con una variedad de herramientas y servicios de terceros.

3. Conceptos Básicos de Git y sus Comandos más Utilizados.

3.1 Descarga e Instalación de Git en Windows

1. **Descargar Git**:
   - Ve al sitio web oficial de Git: https://git-scm.com/
   - Haz clic en el enlace de descarga para Windows para comenzar la descarga.

2. **Ejecutar el instalador**:
   - Una vez que se complete la descarga, haz doble clic en el archivo descargado para ejecutar el instalador de Git.

3. **Configurar el instalador**:
    - Selecciona el idioma que prefieras para la instalación y haz clic en "OK".

4. **Aceptar los términos de la licencia**:
    - Lee los términos de la licencia de Git y, si estás de acuerdo, marca la casilla "I accept the terms in the License Agreement".

5. **Seleccionar componentes**:
    - En la siguiente pantalla, puedes dejar las opciones predeterminadas o personalizar la instalación según tus necesidades. Asegúrate de que "Git Bash Here" esté seleccionado, ya que proporciona una terminal de línea de comandos con Git.

6. **Seleccionar el editor de texto**:
    - En la siguiente pantalla, se te pedirá que elijas un editor de texto predeterminado para Git. Puedes elegir entre Nano, Notepad++ o usar el editor de texto predeterminado de Git. Selecciona tu preferencia y haz clic en "Next".

7. **Configurar el PATH del sistema**:
    - En la pantalla "Adjusting your PATH environment", se te darán tres opciones:
        - "Use Git from the Windows Command Prompt": Esto te permitirá utilizar Git desde el símbolo del sistema de Windows.
        - "Use Git and optional Unix tools from the Command Prompt": Esto te permitirá utilizar Git y herramientas Unix adicionales desde el símbolo del sistema de Windows.
        - "Use Git from Git Bash only": Esto limitará el uso de Git solo a la terminal de Git Bash.
    - Elige la opción que prefieras y haz clic en "Next".

8. **Seleccionar el emulador de terminal** (opcional):
    - Si quieres, puedes elegir entre "MinTTY" o "Windows' default console window" como el emulador de terminal. Selecciona tu preferencia y haz clic en "Next".

9. **Configurar la línea de finalización de los comandos (autocompletado)**:
    - Selecciona si deseas utilizar el "Git Credential Manager" para habilitar la línea de finalización de los comandos. Haz tu selección y haz clic en "Next".

10. **Instalar Git**:
    - Haz clic en el botón "Install" para comenzar la instalación de Git en tu sistema.

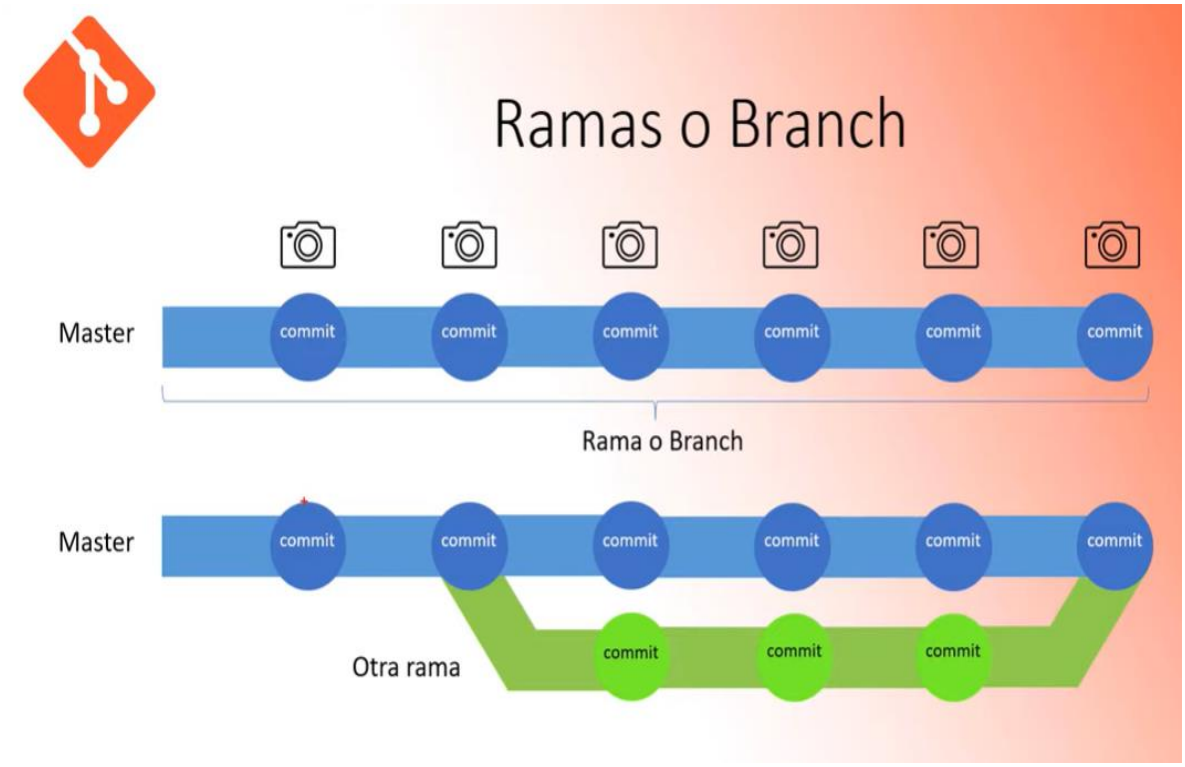
11. **Finalizar la instalación**:
    - Una vez que la instalación haya finalizado, haz clic en "Finish" para cerrar el instalador.

3.2 Conceptos Básicos de Git.

1. **Repositorio (Repository):**
    - Un repositorio Git es un almacén de datos donde se guarda el historial de cambios de un proyecto de software.
    - Puede ser local (en la máquina del usuario) o remoto (alojado en un servidor como GitHub o GitBash).

2. **Commit:**
    - Un commit representa un conjunto de cambios realizados en los archivos del proyecto en un momento específico.
    - Cada commit tiene un mensaje descriptivo que resume los cambios realizados.

3. **Rama (Branch):**
    - Una rama es una línea de desarrollo independiente dentro del repositorio.
    - Permite trabajar en nuevas características o arreglos de errores sin afectar la rama principal del proyecto (generalmente llamada "master" o "main").



4. **Fusión (Merge):**
    - La fusión es el proceso de combinar los cambios de una rama en otra.
    - Se utiliza para incorporar los cambios realizados en una rama de desarrollo a la rama principal del proyecto.

5. **Conflicto de fusión (Merge conflict):**
    - Ocurre cuando Git no puede realizar automáticamente la fusión de cambios debido a conflictos en los archivos.
    - Se debe resolver manualmente, seleccionando qué cambios mantener y qué cambios descartar.

3.3 Comandos más Utilizados en Git.

1. **git init:**
    - Inicializa un nuevo repositorio Git en el directorio actual.

2. **git clone:**
    - Clona un repositorio Git existente en la máquina local.
      git clone <URL del repositorio>

3. **git add:**
    - Agrega cambios de archivos al área de preparación (staging area) para ser incluidos en el próximo commit.
      git add <nombre del archivo>, si es git add . quiere decir que se le hace a todo el proyecto.

4. **git commit:**
    - Crea un nuevo commit con los cambios en el área de preparación.
      git commit -m "Mensaje descriptivo del commit"

5. **git push:**
    - Sube los commits locales a un repositorio remoto.
      git push <nombre del repositorio remoto> <nombre de la rama>

6. **git pull:**
    - Descarga los cambios del repositorio remoto y los fusiona con la rama actual.
      git pull <nombre del repositorio remoto> <nombre de la rama>

7. **git branch:**
    - Lista, crea o elimina ramas.
    - **Crear una rama**: Para crear una nueva rama en Git, puedes utilizar el comando git branch seguido del nombre de la nueva rama que deseas crear. Este comando crea una nueva rama en el repositorio, pero no cambia a esa rama, es decir, sigues trabajando en la rama actual.
      git branch nombre de la rama

- **Listar ramas**: Para ver la lista de todas las ramas en tu repositorio Git, puedes usar el comando git branch. Este comando muestra todas las ramas disponibles en el repositorio, resaltando la rama en la que te encuentras actualmente.
- **Eliminar una rama**: Para eliminar una rama en Git, puedes usar el comando git branch -d seguido del nombre de la rama que deseas eliminar. Este comando elimina la rama especificada. Sin embargo, si hay cambios en la rama que aún no han sido fusionados en otra rama, Git no permitirá la eliminación de la rama por seguridad.
  git branch -d <nombre de la rama>.

8. **git merge:**
   - Fusiona los cambios de una rama en otra.
     git merge <nombre de la rama>
   - Cuando ejecutas **git merge <nombre-de-la-rama>,** Git fusionará los cambios de la rama especificada en la rama en la que te encuentras actualmente. Si no hay conflictos entre los cambios, Git realizará la fusión automáticamente y crearán un nuevo commit de fusión. Sin embargo, si hay conflictos, Git te pedirá que resuelvas esos conflictos manualmente antes de que se pueda completar la fusión.
   - Ejemplo: Supongamos que estás trabajando en una rama llamada **master** y deseas fusionar los cambios de la rama **javascript** en tu rama actual. El siguiente comando realizará la fusión:
     git checkout master
     git merge javascript

9. **git status:**
   - Muestra el estado actual del repositorio, incluyendo archivos modificados, archivos en el área de preparación y ramas.
     git status -s

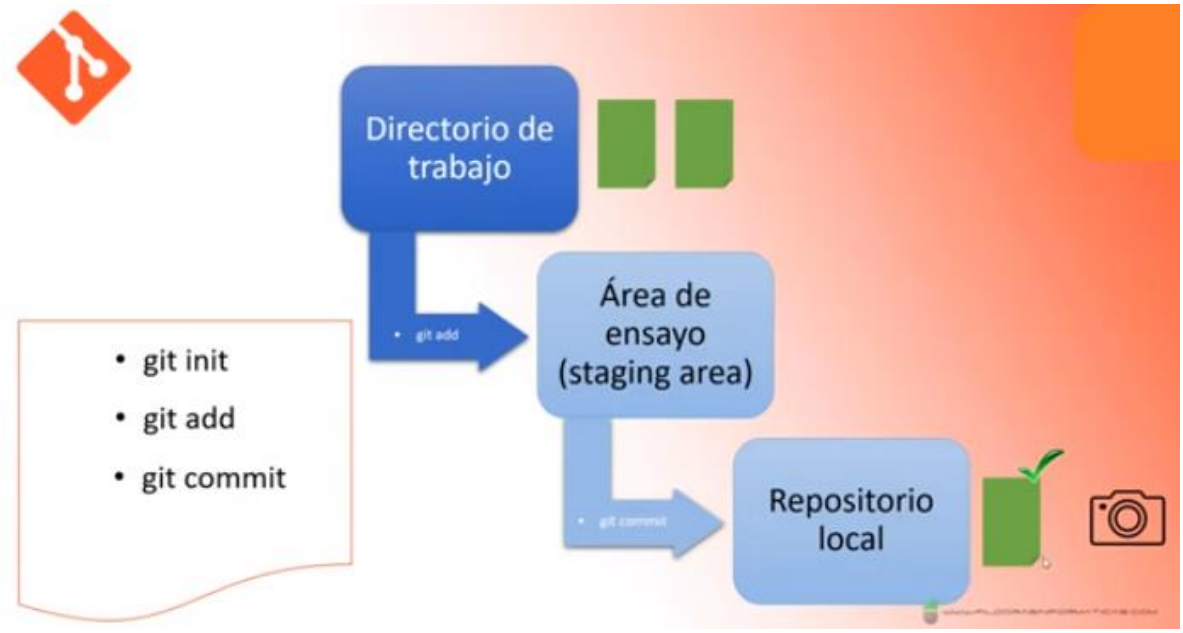
10. **git log –oneline:** Muestra un historial condensado de los commits en una sola línea, lo que facilita la visualización de la historia del repositorio. Cada commit se muestra en una única línea, mostrando su hash de commit (identificador único) seguido del mensaje de commit. Es útil para obtener una visión general rápida de los cambios realizados en el repositorio.

11. **git reset –hard**: Se utiliza para restablecer el estado del repositorio al estado de un commit específico. Con la opción --hard, todos los cambios realizados en el directorio de trabajo y en el área de preparación se descartan y se restauran según el commit especificado, lo que implica una eliminación completa de los cambios no confirmados. Este comando es útil para deshacer cambios no deseados o para volver a un estado anterior del repositorio de forma definitiva.

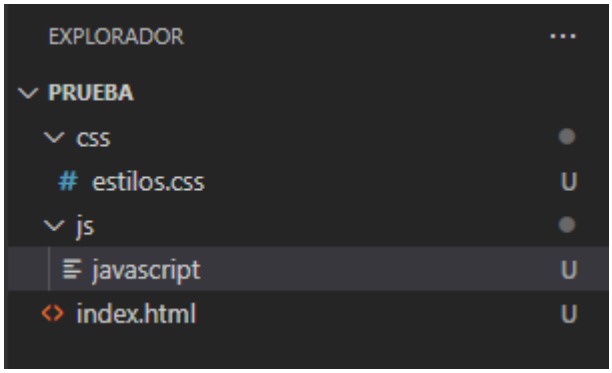
12. **git diff [archivo]** para precisar la muestra. El comando git diff --cached mostrará el contenido que se ha añadido al área de ensayo, es decir, los cambios que se incluirán en el historial si se hace commit.

13. **git checkout:** Para cambiar a una rama específica, puedes utilizar el comando **git checkout** seguido del nombre de la rama. Este comando te permite moverte entre diferentes ramas de tu repositorio.
    git checkout nueva-rama

3.4 Ejemplo Usando Comandos Git.



1. Crea un directorio (prueba) y dentro creamos un directorio css, un directorio js y un archivo index.html, dentro del directorio css creamos un archivo estilos.css, lo mismo hacemos con el directorio js y dentro creamos un archivo javascript.js. Todo esto lo hacemos con el programa Visual Studio Code, luego abrimos la consola con el comando Ctrl + ñ, este comando nos ubica en la ruta indicada.



2. Inicializa un nuevo repositorio Git en el directorio del proyecto: git init.

3. Se verifica el estado de los archivos del proyecto: git status -s.

```
PS C:\Users\ANDRES\Desktop\prueba> git status -s
?? css/
?? index.html
?? js/
PS C:\Users\ANDRES\Desktop\prueba>
```

4. Realiza algunos cambios en los archivos del proyecto y añádelos al área de preparación: git add .

5. Se verifica el estado de los archivos del proyecto: git status -s.

```
PS C:\Users\ANDRES\Desktop\prueba> git add .
PS C:\Users\ANDRES\Desktop\prueba> git status -s
A  css/estilos.css
A  index.html
A  js/javascript
PS C:\Users\ANDRES\Desktop\prueba>
```

6. Crea un nuevo commit con los cambios y proporciona un mensaje descriptivo: git commit -m "Inicio del proyecto".

```
PS C:\Users\ANDRES\Desktop\prueba> git commit -m "Inicio del proyecto"
[master (root-commit) 56b6cd1] Inicio del proyecto
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 css/estilos.css
 create mode 100644 index.html
 create mode 100644 js/javascript
PS C:\Users\ANDRES\Desktop\prueba>
```

Nota. La primera vez que se realiza un commit solicita la información de nombre y correo, utilizando las siguientes instrucciones: git config --global user.email "correo@gmail.com" y git config --global user.name "Nombre". Cada vez que se agregue un commit de un archivo determinado se debe ejecutar primero el comando: git add .

7. Se realizo un cambio en el archivo index.html y se realizo un commit proporcionando un mensaje descriptivo: git commit -m "Primer cambio".

```
index.html > h1
1    <h1>Bienvenidos estudiantes al desarrollo de aplicaciones web</h1>

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  COMENTARIOS

PS C:\Users\ANDRES\Desktop\prueba> git add index.html
PS C:\Users\ANDRES\Desktop\prueba> git commit -m "Primer cambio"
[master 2ae1f19] Primer cambio
 1 file changed, 1 insertion(+)
PS C:\Users\ANDRES\Desktop\prueba>
```

8. Se listan las ramas hasta el momento con el comando **git branch**.

```
PS C:\Users\ANDRES\Desktop\prueba> git branch
* master
PS C:\Users\ANDRES\Desktop\prueba>
```

Luego se crea una rama para cada directorio (css y js).

```
PS C:\Users\ANDRES\Desktop\prueba> git branch estilos
PS C:\Users\ANDRES\Desktop\prueba> git branch javascript
PS C:\Users\ANDRES\Desktop\prueba>
```

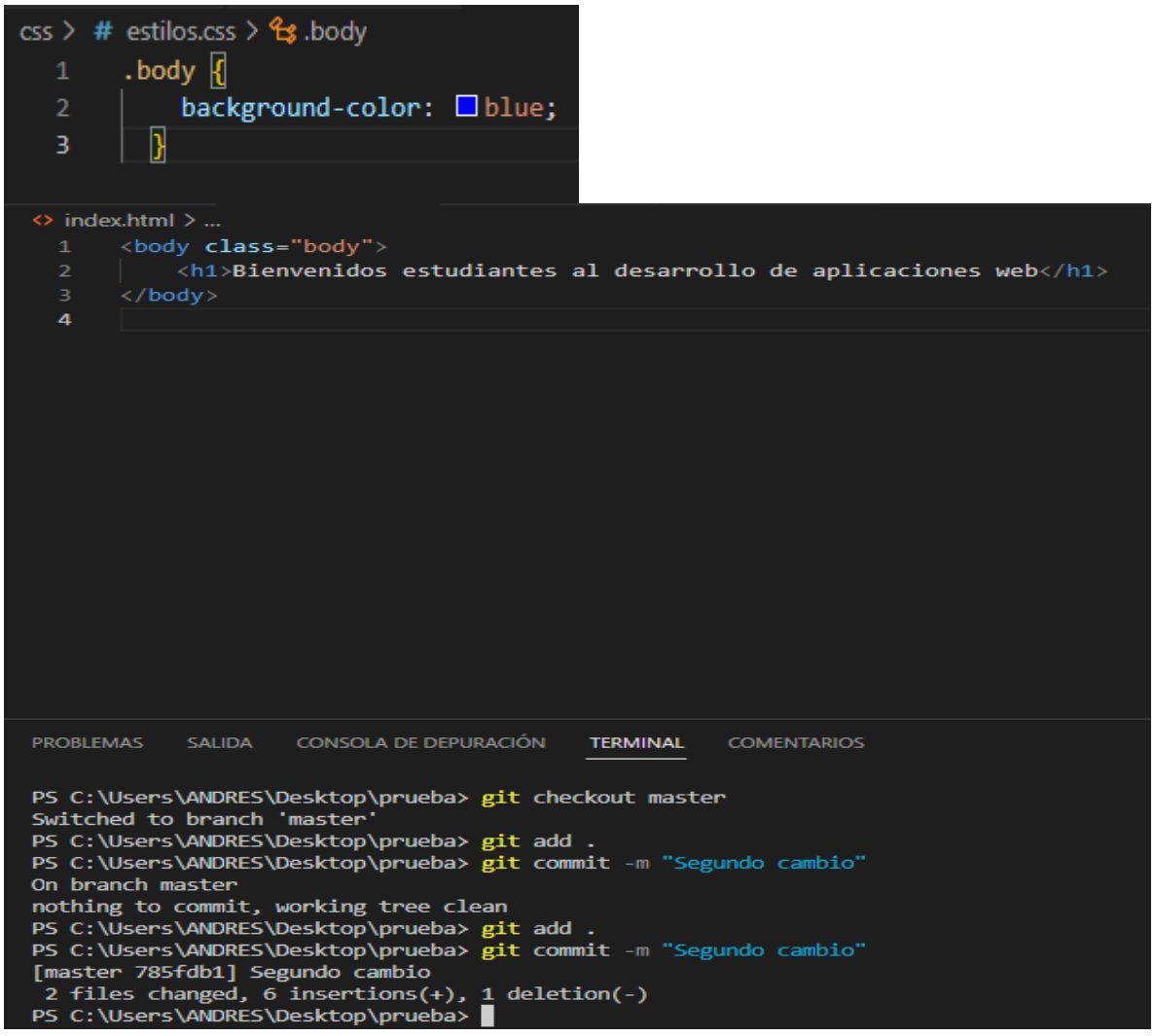
Se listan nuevamente las ramas para visualizar los cambios realizados.

```
PS C:\Users\ANDRES\Desktop\prueba> git branch
  estilos
  javascript
* master
PS C:\Users\ANDRES\Desktop\prueba>
```

Para movernos entre ramas utilizamos el comando git checkout, por ejemplo, para movernos a la rama estilos lo hacemos de la siguiente manera: git checkout estilos

```
PS C:\Users\ANDRES\Desktop\prueba> git checkout estilos
Switched to branch 'estilos'
PS C:\Users\ANDRES\Desktop\prueba>
```

Realizamos un cambio en el archivo index.html y en el archivo estilos.css



Para ver el listado de las copias y su descripción se utiliza el comando git log –oneline.

```
PS C:\Users\ANDRES\Desktop\prueba> git log --oneline
53acdc6 (HEAD -> estilos) Cuarto cambio
2ae1f19 (javascript) Primer cambio
56b6cd1 Inicio del proyecto
PS C:\Users\ANDRES\Desktop\prueba>
```

Para visualizar los cambios realizados a nivel de código en determinado archivo se utiliza el comando git diff nombre del archivo. Para nuestro ejemplo, **git diff 2ae1f19**

```
PS C:\Users\ANDRES\Desktop\prueba> git diff 2ae1f19
diff --git a/css/estilos.css b/css/estilos.css
index e69de29..86bf755 100644
--- a/css/estilos.css
+++ b/css/estilos.css
@@ -0,0 +1,3 @@
+.fondo {
+    background-color: blue;
+ }
\ No newline at end of file
diff --git a/index.html b/index.html
:
```

9. Para fusionar las tres ramas (master, estilos y javascript) en una sola rama se utiliza el comando git merge nombre de la rama, primero debemos cerciorarnos que nos encontramos en la rama **master** (con el comando git checkout master) y luego ejecutamos **git merge estilos y git merge javascript**

```
PS C:\Users\ANDRES\Desktop\prueba> git merge estilos
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Auto-merging css/estilos.css
CONFLICT (content): Merge conflict in css/estilos.css
Automatic merge failed; fix conflicts and then commit the result.
```

```
PS C:\Users\ANDRES\Desktop\prueba> git log --oneline
785fdb1 (HEAD -> master) Segundo cambio
2ae1f19 (javascript) Primer cambio
56b6cd1 Inicio del proyecto
PS C:\Users\ANDRES\Desktop\prueba>
```

10. Para **subir el repositorio desde Visual Studio Code a GitHub**, es necesario tener abierta la cuenta de GitHub y crear un nuevo repositorio en

el icono + opción New Repository, se le asigna un nombre, de tipo public y luego, con las siguientes líneas de código en la consola de Visual Studio Code se sube el proyecto a GitHub:
git remote add origin https://github.com/usuario/prueba.git
git branch -M master
git push -u origin master

11. Para **eliminar un repositorio en GitHub**, sigue estos pasos:

1. Ve a la página principal del repositorio que deseas eliminar en GitHub.

2. Haz clic en el botón "Settings" (Configuración) en la esquina superior derecha del repositorio.

3. Desplázate hacia abajo hasta la sección "Danger Zone" (Zona de peligro).

4. Haz clic en el enlace "Delete this repository" (Eliminar este repositorio).

5. Te pedirán que escribas el nombre del repositorio para confirmar la eliminación. Escribe el nombre del repositorio tal como se muestra.

6. Haz clic en el botón rojo "I understand the consequences, delete this repository" (Entiendo las consecuencias, eliminar este repositorio).

   Una vez que confirmes la eliminación, el repositorio y todos sus datos asociados serán eliminados permanentemente. Ten en cuenta que esta acción no se puede deshacer, así que asegúrate de que realmente quieres eliminar el repositorio antes de proceder.

5. GitHub Pages

GitHub Pages es un servicio de alojamiento de sitios web estáticos que permite a los usuarios publicar sitios web directamente desde sus repositorios de GitHub. Es una excelente manera de compartir documentación, proyectos personales, blogs, sitios web estáticos y más.

5.1 Características.

1. **Fácil de usar:** GitHub Pages es fácil de configurar y usar. Simplemente puedes alojar tu sitio web estático directamente desde un repositorio de GitHub.

2. **Integración con Git:** Como GitHub Pages utiliza Git para el control de versiones, puedes gestionar tu sitio web utilizando Git como lo harías con cualquier otro proyecto.

3. **Soporte para dominios personalizados:** Puedes asociar tu propio nombre de dominio con tu sitio web alojado en GitHub Pages, lo que te permite tener una URL personalizada para tu sitio.

4. **Plantillas predefinidas:** GitHub Pages ofrece una serie de plantillas predefinidas que puedes utilizar para configurar rápidamente tu sitio web.

5. **Totalmente gratuito:** GitHub Pages es un servicio gratuito para todos los usuarios de GitHub, lo que lo convierte en una opción económica para alojar sitios web personales y proyectos de código abierto.

5.2 Cómo Utilizar GitHub Pages.

Ya con el repositorio creado en GitHub y deseas crear una página web utilizando la función de GitHub Pages, puedes seguir estos pasos:

1. Ve a la página de tu repositorio en GitHub.

2. Haz clic en la pestaña "Settings" (Configuración) en la parte superior derecha del repositorio.

3. Desplázate hacia abajo hasta encontrar la sección "Pages".

4. En la opción "Source" (Fuente), selecciona la rama que contiene tu código fuente. Por lo general, esto sería la rama 'main' o 'master'.

5. Selecciona la carpeta de tu proyecto que contiene los archivos HTML, CSS, JavaScript, u otros archivos necesarios para tu sitio web. Si tu proyecto está en la raíz del repositorio, puedes seleccionar "/root".

6. Haz clic en "Save" (Guardar).

GitHub Pages ahora comenzará a construir tu sitio web a partir de los archivos seleccionados en la rama especificada. Una vez que la construcción esté completa, podrás acceder a tu sitio web utilizando la URL proporcionada en la sección "GitHub Pages" de la configuración de tu repositorio. Esta URL generalmente seguirá el formato **https://<usuario>.github.io/<repositorio>**.

Asegúrate de que tu proyecto tenga una estructura de archivos adecuada para una página web y que incluya un archivo HTML principal, así como cualquier otro recurso necesario (CSS, JavaScript, imágenes, etc.).