ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS
INTRODUCCIÓN AL DESARROLLO WEB
2025 II
LABORATORIO 02 – HERRAMIENTAS DEL DESARROLLADOR WEB

I

OBJETIVOS

- Utilizar el manejo de versiones con Git y GitHub
- Explorar las principales herramientas del desarrollador web en diferentes sistemas operativos
- Solucionar ejercicios aplicando las herramientas vistas

TIEMPO ESTIMADO: 2 horas

II

CONSIDERACIONES DE EVALUACIÓN

- Se deberán utilizar los conocimientos impartidos en las clases teóricas
- Deberá utilizar nombre de variables significativos
- Deberá realizar pruebas adicionales
- El alumno deberá indicar en su código con quien colaboró, así sea la IA
- El alumno será requerido de realizar modificaciones en su código y responder a preguntas sobre el mismo
- Los ejercicios deberán realizarse en el laboratorio, a su ritmo y subirlos al aula virtual como AVANCE antes de finalizar la clase
- Todos los ejercicios completos, deberán ser subidos al aula virtual como TAREA, para lo cual se les dará un tiempo adecuado y deben cumplir el deadline estipulado. Esto se deberá cumplir, aunque en el laboratorio ya se hayan terminado todos los ejercicios
- El formato a usar para los avances y tareas será .zip, que contenga todos los archivos requeridos
- Utilizar Git con GitHub para el manejo de versiones de su trabajo, ocasionalmente se le solicitará que compartan sus repositorios

III

POLITICA DE COLABORACION

La política del curso es simple, a menos que se exprese lo contrario en el laboratorio, siéntase libre de colaborar con sus compañeros en todos los laboratorios, pero debe notificar expresamente con quien ha colaborado. La colaboración con alumnos, que no están matriculados en el curso está prohibida. Los laboratorios y asignaciones han sido desarrollados para ayudarlo a comprender el material. Conozca su código y esté preparado para revisiones individuales de código. Durante las revisiones es probable que se le pida realizar modificaciones y justificar sus decisiones de programación. Cada uno de sus ejercicios debe iniciar de la siguiente forma:

```
// Laboratorio  Nro x - Ejerciciox
// Autor: mi nombre
// Colaboró : el nombre
// Tiempo :
```

INDICACIONES GENERALES

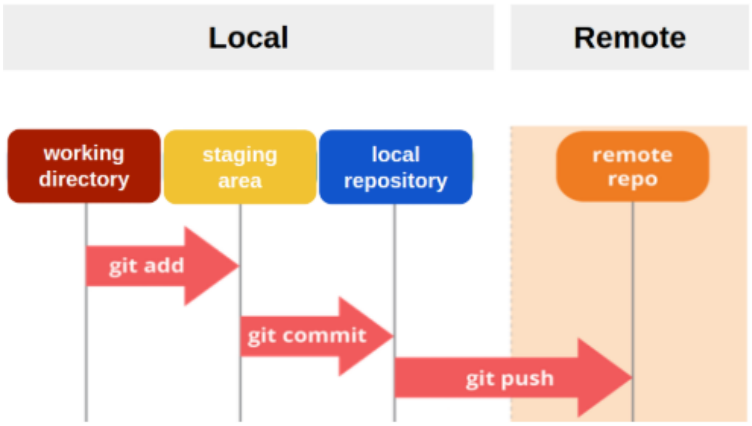
a. En cada sesión de laboratorio, los ejercicios propuestos deberán ser guardados en la misma carpeta/directorio
b. La carpeta deberá tener el nombre del Laboratorio y el nombre del alumno, así por ejemplo:
Laboratorio 11 – Juan Perez
c. Utilice nombres significativos
d. Su código deberá estar correctamente indentado y preferentemente documentado
e. Deberá ser debidamente probado

MARCO TEORICO

1. Git y GitHub

Conceptos clave:

- Repositorio local: la copia del proyecto que está en tu computadora
- Repositorio remoto: la copia en un servidor (ejemplo: GitHub)
- Commit: Un "punto de guardado" en el historial del proyecto (foto instantánea)
- Branch (rama): Una línea de desarrollo independiente
- Push/Pull: Enviar y traer cambios entre repositorios locales y remotos



Procedimiento:

Puede realizar la actividad en la terminal Git Bash de Visual Studio Code para Windows o Linux, o en la terminal de Linux.

A) Configuración inicial

1. Configurar Git con tu usuario: es necesario hacerlo **sólo la primera vez** para configurar Git en tu PC

```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@ejemplo.com"
```

2. Verificar configuración: (hacia el final de la lista se pueden ver los 2 datos anteriores)

```
git config --list
```

3. Ver el estado del repositorio

```
git status
```

B) Crear un repositorio local

1. Crear carpeta y entrar en ella

```
mkdir laboratorio-git
cd laboratorio-git
```

2. Inicializar Git: sólo se hace una vez para cada repositorio

```
git init
```

Opcional

```
git branch -M main
```

a veces la rama principal tiene el nombre de “master”, se cambia a “main”

3. Crear archivo `index.html`, ingresar algún contenido inicial

4. Agregar el archivo al área de preparación (staging):

```
git add index.html
```

o para añadir todos los archivos

```
git add .
```

5. Guardar cambios con commit:

```
git commit -m "Primer commit - agregar index"
```

6. Ver historial:

```
git log --oneline
git log
```

7. Añadir el archivo `styles.css` y repetir 4, 5, 6. En paso 5 cambiar comentario

8. Volver a una versión anterior mueve la rama actual (por ejemplo: main) para que ahora apunte a ese commit

```
git reset --hard commitID
```

(commitID es el hash del commit, verlo con git log)

Borra (en esa rama) todos los commits que venían después

9. Volver a una versión anterior modo lectura (para mirar o probar cosas)

```
git checkout commitID
```

(commitID es el hash del commit, verlo con git log)

Si quieres seguir trabajando desde el commit donde estabas antes:

```
git checkout -
git checkout nombre_rama
```

Si quieres seguir trabajando desde ahí, debes crear una nueva rama sin perder cambios

```
git checkout -b nombre_rama
```

Ejemplo 1: repositorio con un archivo y crea 3 commits. Crear una carpeta ejemplogit01 y abrirla desde VS Code o Terminal

```
git init
# crear index.html
git add index.html
git commit -m "Commit 1: versión inicial de index.html"

# haces cambios en index.html...
git add index.html
git commit -m "Commit 2: agregando contenido html"

# haces más cambios...
git status
git add index.html
git status
git commit -m "Commit 3: agregando html semantico"

git log --oneline
```

Ejemplo 2: repositorio con un archivo y crea 3 commits. Crear una carpeta ejemplogit02 y abrirla desde VS Code o Terminal

```
git init
# crear index.html
git add index.html
git commit -m "Commit 1: versión inicial de index.html"

# crear styles.css
git add styles.css
git commit -m "Commit 2: agregando estilos"

# crear script.js
git add script.js
git commit -m "Commit 3: agregando script JS"

git log --oneline
```

a1b2c3d (HEAD -> main) Commit 3: agregando script JS
b2c3d4f Commit 2: agregando estilos
a3c4d5e Commit 1: versión inicial de index.html

En Git siempre existen dos punteros importantes que ayudan a ubicar en qué parte del historial estás trabajando:

- HEAD: indica dónde estás trabajando (apunta a una rama)
- Rama (branch pointer): apunta al último commit de dicha rama

a3c4d5e -> b2c3d4f -> a1b2c3d (main)
↑
HEAD

Ejemplo 3: siguiendo con el anterior ejemplo, si quiero volver al segundo commit:

```
# Volver a un commit anterior
# Primera ruta
git checkout b2c3d4f

# Si quieres volver al último commit donde estabas:
git checkout main            o master, si así se llama la rama principal

# Si quieres seguir trabajando desde ahí, crea una nueva rama:
git checkout -b desde-commit-2


# Segunda ruta
git reset --hard b2c3d4f
# La rama main ahora apunta al commit 2
# El commit 3 ya no está en main
```

C) Conectar con GitHub

Necesita tener una cuenta en GitHub.

1. Crear repositorio en GitHub New – Nombre y Descripción – Create Repository

2. Copiar el url del repositorio Code - Copiar

3. Vincular remoto: desde la terminal

```
git branch -M main
```
a veces la rama principal tiene el nombre de “master”, se cambia a “main”

```
git remote add origin https://github.com/usuario/laboratorio-git.git
```
registrar una conexión remota entre tu repositorio local y el remoto ubicado en dicho URL, a partir de ahora origin será el alias de dicho URL

4. Subir cambios:

```
git push -u origin main
```
sube la rama main al remoto origin (sólo esa rama)
sólo la primera vez, después se puede usar git push o git pull

```
git remote -v
```
para ver con que URL está asociado origin, para enviar (push) y traer (fetch/pull) cambios

D) Clonar y sincronizar: en caso se quiera traer un repositorio a tu pc

1. Clonar un repositorio: sólo se hace la primera vez

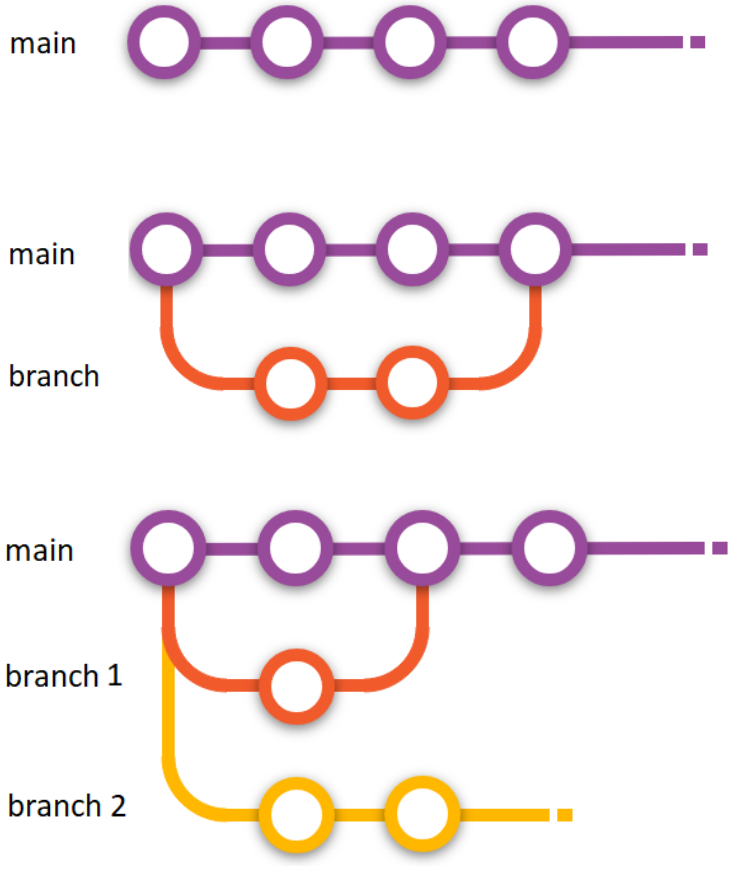
```
git clone https://github.com/usuario/laboratorio-git.git
```

2. Traer cambios del remoto: las siguientes veces para actualizar la copia local

```
git pull origin main
```
tener cuidado desde donde se llama

E) Uso de ramas

Una rama (branch) es una línea de desarrollo independiente dentro de un repositorio.
La rama principal suele ser main (moderno) o master.
Con ramas puedes experimentar o desarrollar una nueva funcionalidad sin afectar el código estable.



1. Crear nueva rama:

```
git branch nombre-de-rama          crea rama
git checkout nombre-de-rama        cambia a esa rama
git switch nombre-de-rama          forma moderna de la anterior
git checkout -b nombre-de-rama     crea rama y se cambia a ella en un solo paso
git switch -c nombre-de-rama       forma moderna de la anterior
git branch                         ve en qué rama estás
```

2. Editar archivo `index.html` y guardar cambios

3. Confirmar cambios:

```
git add .
git commit -m "Agregada nuevos elementos al index.html"
```

4. Subir rama al remoto:

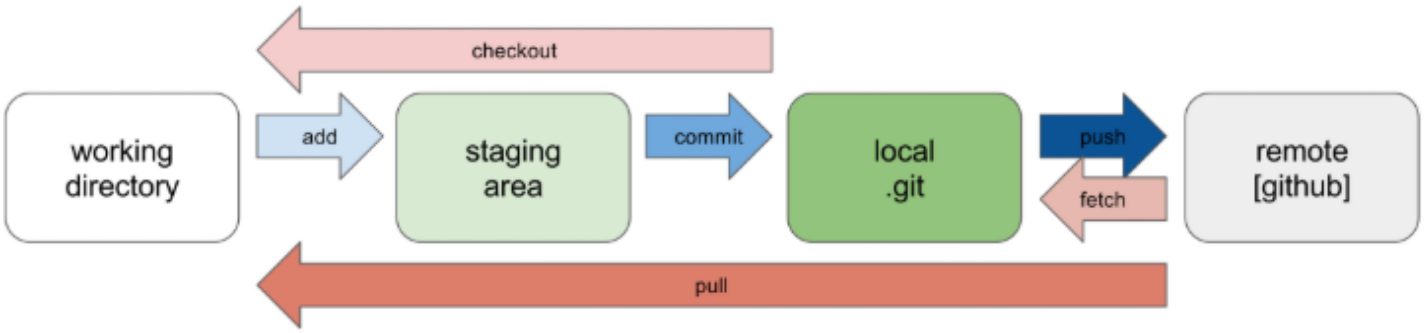
```
git push -u origin nombre-de-rama        sube la rama nueva
git push --all origin                    subir todas las ramas locales
```

5. Muestra todas las ramas, puedes ver merges, subramas, y cómo se conectan los commits

```
git log --oneline --graph --all
```

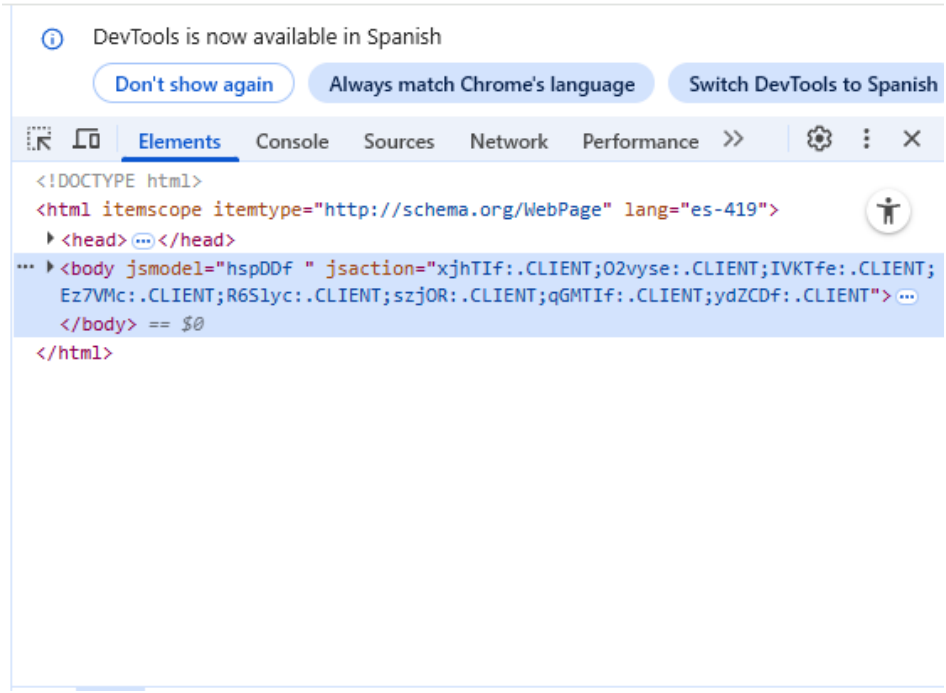
Secuencia típica

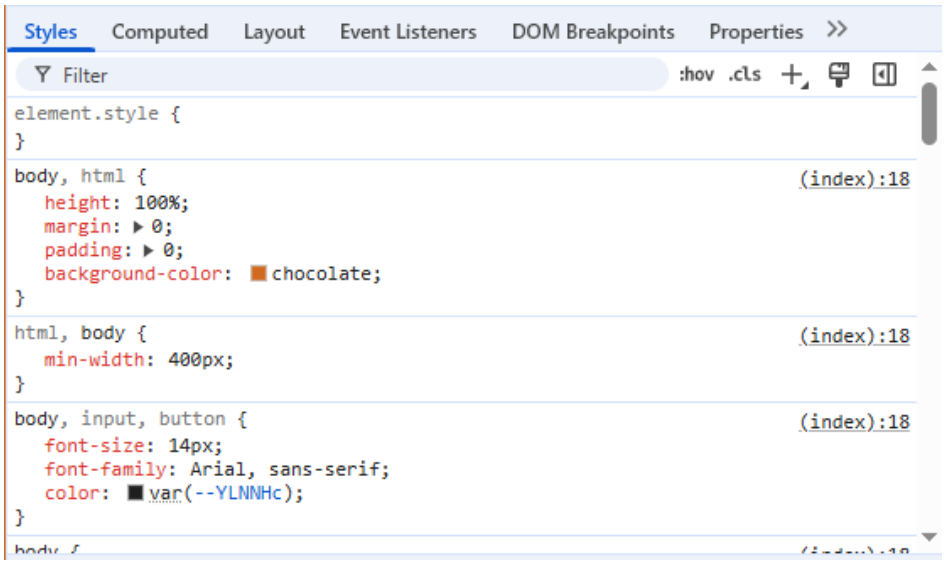
```
# asegurarse de estar en rama main
# crear rama y cambiarse a ella
git switch -c nombre-rama-nueva     # git checkout -b nombre-rama-nueva
# hacer cambios...
git add .
git commit -m "Agregados los cambios…"
# ir a main
git checkout main
# fusionar o unir
git merge nombre-rama-nueva
# si quisieras borrar la rama
git branch -d nombre-rama-nueva


# si quisieras subir rama a remoto
git push -u origin nombre-rama-nueva

# en remoto, tú u otro desarrrollador puede hacer cambios
# Opción 1: traer a local y fusionar directamente
git pull origin nombre-rama-nueva
# Opción 2: traer a local sin fusionar    , se puede revisar y luego fusiona
git fetch origin
git merge origin/nombre-rama-nueva
```



2. DevTools (Herramientas del Desarrollador) en Linux

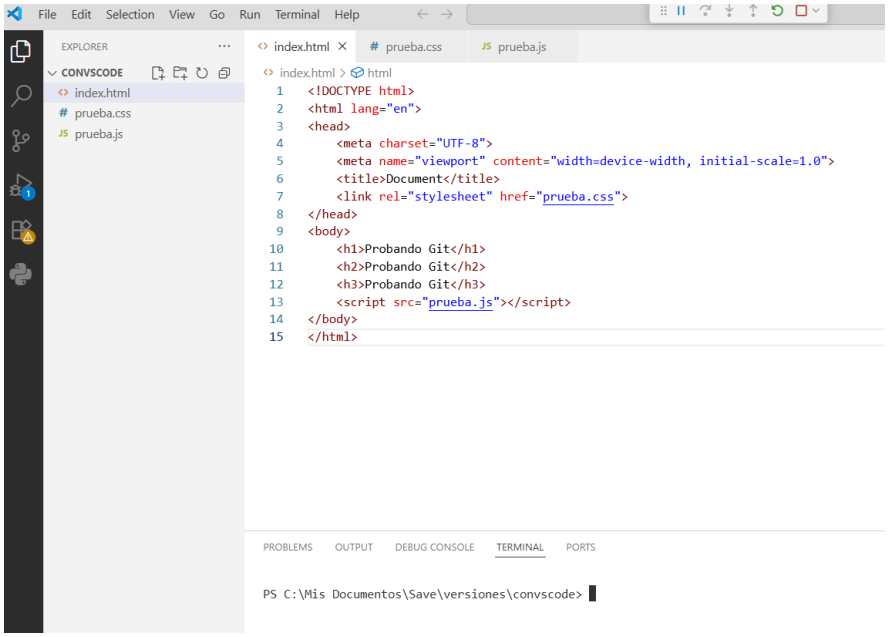
2.1. Abrir un browser y comprobar la utilización de DevTools en una distribución de Linux

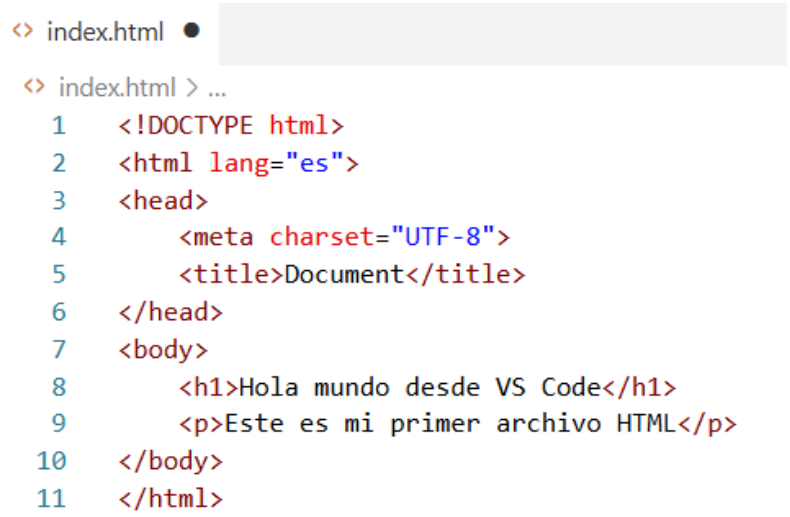


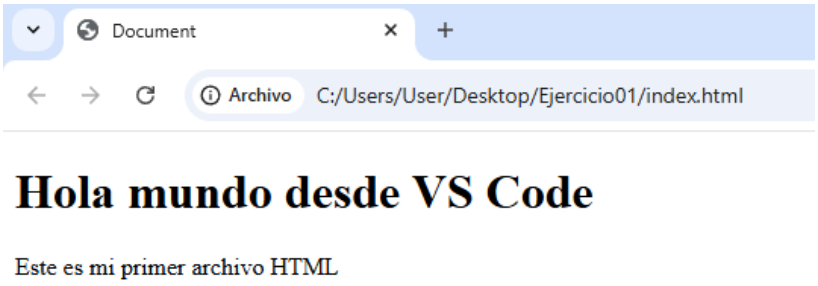


3. Visual Studio Code

3.1. Comprobar su utilización en una distribución de Linux







4. **Terminal con VS Code**
   4.1. Abrir con Ctrl + `. Algunos atajos de teclado varían.
   4.2. Usar Git Bash (necesita instalar Git)

5. **Terminal de Linux**

Comando	Descripción
De directorio	
`pwd`	Muestra la ruta actual
`ls`	Lista archivos y directorios
`cd nombre_dir`	Cambia de directorio
`cd ..`	Subir un nivel en la ruta
`cd ~`	Ir al home del usuario
`mkdir nombre_dir`	Crea un directorio
`rm -r nombre_dir`	Elimina un directorio con contenido
`rmdir nombre_dir`	Elimina un directorio vació
De archivo	
`touch archivo.txt`	Crea un archivo
`cat archivo.txt`	Muestra el contenido
`rm archivo.txt`	Borra un archivo (pueden ser varios)
`cp origen destino`	copiar archivo
`cp -r carpeta destino`	copiar directorio con todo
`mv origen destino`	mover o renombrar archivo/directorio
`less archivo.txt`	permite leer con desplazamiento
`head archivo.txt`	muestra primeras 10 líneas
`tail archivo.txt`	muestra últimas 10 líneas
`echo "texto" > archivo.txt`	Sobrescribe un archivo
`echo "texto" >> archivo.txt`	Agrega texto al final del archivo
Otros	
`clear` ctrl+L	Limpia la terminal
`xdg-open .`	Abre el explorador de archivos en la carpeta actual
`xdg-open index.html`	Abre un archivo con el programa asociado (browser para html)

EJERCICIOS PROPUESTOS

1. Usando VS Code, en una carpeta del escritorio que se llame ejercicio01, crear 3 archivos: index.html, styles.css, prueba.js. En el html que muestre como cabecera principal (h1) su nombre completo y como párrafo (p) una descripción de su experiencia en el primer semestre de la carrera profesional. Capturar pantalla que demuestre los resultados.

2. En la terminal de Linux, en el directorio de su usuario, crear una nueva carpeta ejercicio02, dentro crear 3 archivos: index.html, styles.css, prueba.js. En el html que muestre como cabecera principal (h2) su nombre completo y como párrafo (p) una descripción de su experiencia en el primer semestre de la carrera profesional.
   En el css escribir: p{color:rojo}.
   En el js escribir alert("hola").
   Copiar el index.html con su mismo nombre, a un subdirectorio llamado backup. Verificar que lo copió de forma correcta. Todo usando solamente comandos de la terminal. Capturar pantalla con los comandos usados y sus resultados.

3. Crear un repositorio en GitHub llamado Laboratorio2IDWeb

4. Todo desde la terminal de Linux. Crear la carpeta ejercicio03
   Crear el index.html y crear el primer commit
   Subir al repositorio creado en GitHub
   Crear el estilos.css y crear el segundo commit
   Subir al repositorio creado en GitHub
   Crear el prueba.html y crear el tercer commit
   Subir al repositorio creado en GitHub
   Aumentar un nuevo párrafo en index.html
   Subir al repositorio creado en GitHub

5. Crear una nueva rama y 3 commits en ella. Subir a GitHub. Indicar cómo hacer merge del main con la nueva rama en local y en remoto

6. ¿Qué significa hacer Pull Request en GitHub?

7. ¿Cómo se añaden colaboradores a un repositorio en GitHub?

8. ¿Cómo se resuelven los conflictos que surgen al hacer un merge en un repositorio cuando todos los integrantes del equipo de desarrollo trabajan en sus ramas propias?. Indicar los pasos

I. TAREA PARA LA CASA: Complete todos los ejercicios.

1) Envíen la URL del repositorio público que crearon con GitHub (Ejercicio 9)
2) Trabajo en equipo con Git y GitHub (Ejercicio 10)

Un equipo de 3 desarrolladores va a colaborar en el desarrollo de un proyecto en GitHub. El proyecto tendrá 3 áreas de trabajo:

- Persona A (Frontend): debe trabajar en los archivos index.html y styles.css.
- Persona B (Lógica): debe trabajar en script.js.
- Persona C (Backend): debe trabajar en config.json y db.js.

El repositorio principal tendrá la rama main, que siempre debe estar estable. Nadie debe trabajar directamente en main.

Actividades

1. Preparación del repositorio
   - Una persona del equipo (el líder) crea un repositorio en GitHub llamado proyecto-web1
   - Inicializa la rama main y agrega a los demás desarrolladores del equipo como colaboradores
   - Todos los miembros clonan el repositorio en sus computadoras
2. Creación de ramas de trabajo
   - Cada desarrollador crea una rama propia desde main:
     - Desarrollador A → feature/ui
     - Desarrollador B → feature/logic
     - Desarrollador C → feature/config
3. Trabajo en local
   - Cada desarrollador edita únicamente los archivos que le corresponden en la rama creada
   - Guardan sus cambios, hacen git add, git commit y suben sus ramas al remoto con git push
4. Integración con Pull Requests
   - Cada desarrollador, al terminar su tarea, abre un Pull Request en GitHub hacia la rama main
   - Otro compañero revisa y aprueba los cambios
   - El PR se hace merge en GitHub
5. Sincronización del equipo
   - Después de cada merge, todos los desarrolladores deben actualizar su repositorio local con:
     git checkout main
     git pull origin main
6. Finalización
   - Una vez que los 3 Pull Requests han sido integrados, el proyecto final estará en la rama main, conteniendo el trabajo de todos

Entregables:

- Indicar los comandos de Git usados por cada desarrollador del equipo
- Captura de pantalla del repositorio en GitHub mostrando al menos 3 Pull Requests cerrados
- Captura de pantalla del historial de commits en la rama main
- Envíar la URL del repositorio público que crearon con GitHub para este ejercicio. Ahí estarán todos los archivos del proyecto actualizados en la rama main después de integrar todas las ramas

Crear un documento docx/pdf con la solución de los 10 ejercicios.
Subir el documento a la tarea **Tarea 02** del Aula Virtual respetando las fechas indicadas.