**Informe del Laboratorio 01**

|  |
| --- |
| Nota |
|  |

**Tema 1: Git y GitHub**

|  |  |  |
| --- | --- | --- |
| **Estudiante** | **Escuela** | **Asignatura** |
| Rodrigo E. Infanzón Acosta  [rinfanzona@ulasalle.edu.pe](mailto:rinfanzona@ulasalle.edu.pe) | Carrera Profesional de Ingeniería de Software | Lenguaje de Programación 3  **Semestre IV** |

|  |  |  |
| --- | --- | --- |
| **Laboratorio** | **Tema** | **Duración** |
| 01 | **Git y GitHub** | 06 horas académicas |

|  |  |  |
| --- | --- | --- |
| **Semestre académico** | **Fecha de inicio** | **Fecha de entrega** |
| 2024 - A | 21/03/24 | 27/03/24 |

**Actividades a realizar:**

1. Elaborar un informe individual, acerca de las plataformas de control de versiones: Git y GitHub.
2. Elaborar la solución y explicación de la creación de un tablero M x N, asimismo, la actualización del tablero.cpp usando el primer apellido del estudiante o una "X" en la diagonal.
3. Elaborar la solución al problema usando Git y GitHub, seleccionado por el profesor, denominado “**11451. El perrito que quiere un hueso**”, extraído de la plataforma **omegaUp:** <https://omegaup.com/arena/problem/El-perrito-que-quiere-un-hueso/>
4. Mencionar **tres aportes** a su adquisición de conocimientos que este laboratorio le proporcionó.
5. Utilizar todas las recomendaciones dadas por el docente en clase, tales como:

* **Antecedentes:** describir antecedentes previos que sean necesarios para desarrollar el laboratorio. Las entregadas por el docente y/o las que se buscaron personalmente.
* **Commits:** elaborar la lista de envíos que permitirán culminar el laboratorio, previo a la implementación.
* **Source:** explicar porciones de código fuente importantes, trascendentales que permitieron resolver el laboratorio y que reflejen su particularidad única, sólo en trabajos grupales se permite duplicidad.
* **Ejecución:** muestra comandos, capturas de pantalla, explicando la forma de replicar y ejecutar el entregable del laboratorio.

1. Entregables:

* URL al directorio específico del laboratorio en su repositorio GitHub privado donde esté todo el código fuente y otros que sean necesarios. Evitar la presencia

de archivos: binarios, objetos, archivos temporales. Incluir archivos de especificación como: packages.json, requirements.txt o README.md.

* No olvidar que el docente debe ser siempre colaborador a su repositorio que debe ser privado. (Usuario del docente: **@rescobedoulasalle**).
* Se debe de describir sólo los commits más importantes que marcaron hitos en su trabajo, adjuntando capturas de pantalla, del commit, porciones de código fuente, evidencia de sus ejecuciones y pruebas.
* En el informe siempre se debe explicar las imágenes (código fuente, capturas de pantalla, commits, ejecuciones, pruebas, etc.) con descripciones puntuales pero precisas.
* Agregar la estructura de directorios y archivos de su laboratorio.

**Recursos y herramientas utilizados:**

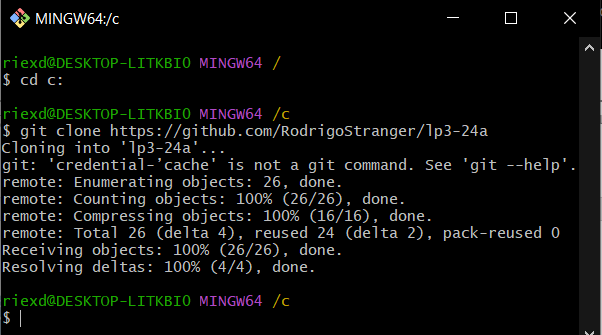
1. Sistema operativo utilizado: Windows 10 pro 22H2 de 64 bits (SO. 19045.4170).
2. Hardware: Ryzen 5 3550H 2.10 GHz, RAM 16 GB DDR4 2400 MHz.
3. Visual Studio Code (versión 1.87.2).
4. Git (versión 2.44.0).
5. Cuenta de GitHub creada con el correo institucional proporcionado por la Universidad La Salle de Arequipa ([rinfanzona@ulasalle.du.pe](mailto:rinfanzona@ulasalle.du.pe)).
6. Conocimientos básicos en Git.
7. Conocimientos básicos sobre programación.

**Información del repositorio en GitHub:**

1. Enlace del repositorio en GitHub: <https://github.com/RodrigoStranger/lp3-24a>
2. Enlace del repositorio para el laboratorio 01 en GitHub: <https://github.com/RodrigoStranger/lp3-24a/tree/main/lab01>

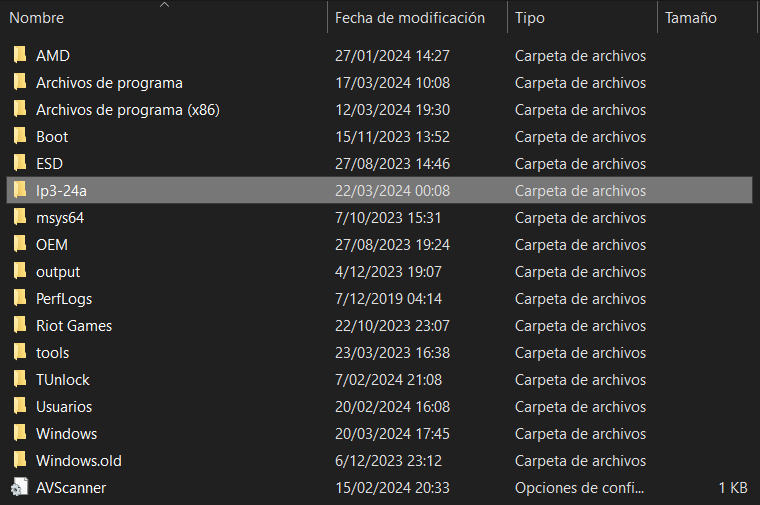
**Actividades realizadas con el repositorio de GitHub en Local:**

1. Como previamente en el laboratorio 15/03/24 realizamos la creación de nuestro repositorio, configurándolo en consola, añadiendo de colaborador a nuestro docente, configurando nuestro token y resolviendo un problema de programación, en la cual pusimos en práctica el uso básico del Git y GitHub, por lo cual, en nuestro equipo local personal, tendríamos que poner en práctica el uso del servidor GitHub, para clonar nuestro repositorio existente en la nube, hacia nuestro equipo personal. Por ende, se realiza de la siguiente manera:
2. **Seleccionar un sistema de comandos:** utilizo la terminal de comandos establecida por Git, denominada **Git Bash**, con ruta en *C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Git*
3. **Determinar en qué directorio se va a clonar el repositorio:** debemos especificar en que directorio de nuestro disco local queremos almacenar nuestro repositorio, en mi caso: *C:\.* Elegido el directorio en la consola, se debe ejecutar el comando: **git clone** <https://github.com/RodrigoStranger/lp3-24a> :



**Imagen con propiedad del autor ©Rodrigo Infanzon Acosta**

1. Producto del resultado en la consola de **Git Bash**, el repositorio clonado, lp3-24a existe en nuestro equipo local, listo para trabajar según las metas establecidas por nuestro docente:



**Imagen con propiedad del autor ©Rodrigo Infanzon Acosta**

**Desarrollo de las actividades:**

**Informe Individual: Git y GitHub**

**Git**

El sistema de control de versiones gratuito y de código abierto, Git, fue originalmente creado por Linus Torvalds en 2005. A diferencia de los sistemas centralizados de control de versiones anteriores, como SVN y CVS, Git adopta un enfoque distribuido con determinadas características:

**Beneficios y Significado:**

Git ha tenido un impacto significativo en el desarrollo de software y la cultura de la colaboración. Algunos de sus beneficios más destacados incluyen:

* **Historial completo de cambios:** Git proporciona un registro detallado de todos los cambios realizados en un proyecto, lo que facilita la auditoría y la reversión de cambios si es necesario.
* **Colaboración eficiente:** la naturaleza distribuida de Git permite a los desarrolladores trabajar de manera independiente y luego integrar sus cambios de manera fluida, lo que facilita la colaboración en equipos distribuidos.
* **Experimentación segura**: con ramas ligeras y fáciles de crear, Git fomenta la experimentación y el desarrollo iterativo de nuevas características sin afectar la estabilidad del proyecto principal.
* **Respaldo y restauración:** Git permite a los desarrolladores realizar copias de seguridad regulares del repositorio y restaurar versiones anteriores en caso de pérdida de datos o errores críticos.

Desde su creación en 2005, Git ha pasado de ser una herramienta utilizada principalmente para el desarrollo del kernel de Linux a convertirse en el estándar de facto para el control de versiones en la industria del software. Su impacto en la eficiencia, la colaboración y la gestión de proyectos es innegable, y su legado continuará influyendo en la forma en que se desarrolla el software en el futuro.

**Configuración:**

**Configurar nuestra identidad global en Git:**

Para configurar tu identidad global en Git, incluyendo tu nombre de usuario y dirección de correo electrónico, sigue estos pasos:

* **git config --global user.name "Tu Nombre"**
* **git config --global user.email tu@email.com**

**Verificar la configuración:** Puedes verificar que la configuración se haya realizado correctamente ejecutando los siguientes comandos:

* **git config --global user.name**
* **git config --global user.email**

**Operaciones básicas de gestión de archivos y carpetas:**

Antes de inicializar un repositorio Git, puedes realizar operaciones básicas en tu sistema de archivos. Algunos comandos comunes necesarios antes de crear un repositorio:

* Elección de disco local: **cd (nombre del disco):**
* Entrar a un directorio: **cd (nombre del directorio)**
* Retroceder de directorio: **cd ..**
* Crear una carpeta: **mkdir (nombre de la carpeta)**
* Eliminar una carpeta: **rmdir (nombre de la carpeta)**
* Leer un archivo: **ls**

**Inicializar un repositorio Git:**

Para comenzar a utilizar Git en un directorio existente, primero debemos inicializar un repositorio:

* **git init**

**Cambiar el nombre de la rama principal:**

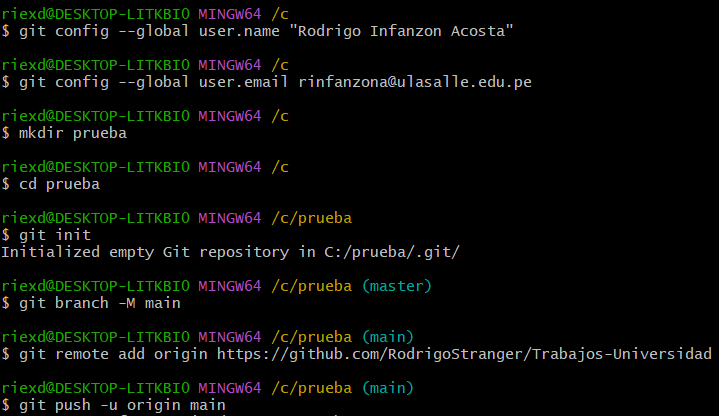
Este paso es opcional, pero si deseamos cambiar el nombre de la rama principal a "main", puedes usar el siguiente comando:

* **git branch -M main**

**Agregar un control remoto:**

A continuación, necesitamos agregar un control remoto a tu repositorio local. Esto generalmente se hace para vincular tu repositorio local con un repositorio remoto, como uno alojado en GitHub. Puedes hacerlo con el comando:

* **git remote add origin** [**https://github.com/RodrigoStranger/Trabajos-Universidad**](https://github.com/RodrigoStranger/Trabajos-Universidad)

**Ejemplo:**

**Imagen con propiedad del autor ©Rodrigo Infanzon Acosta**

**Agregar archivos al área de preparación:**

Antes de hacer un commit, necesitamos agregar los archivos que deseas incluir en el commit al área de preparación o staging area. Puedes hacer esto utilizando el comando git add. Por ejemplo, podemos usar:

* Agregar todos los archivos: **git add .**
* Agregar un archivo: **git add (nombre del archivo.extensión)**
* Eliminar un archivo del area de preparación: **git rm --cached (nombre del archivo)**

**Hacer un commit:**

Después de agregar los archivos al área de trabajo, estás listo para hacer un commit. Un commit en Git representa un conjunto de cambios que se van a guardar en el historial del repositorio. Puedes hacer un commit con el comando:

* **git commit -m “mensaje de confirmación del commit”**
* Hacer un commit sin hacer un git add: **git commit -a**

**Empujar cambios al repositorio remoto:**

Una vez que has agregado el control remoto, puedes empujar tus cambios locales al repositorio remoto usando el comando git push. Para establecer la rama remota de seguimiento y empujar los cambios, puedes usar la opción -u (upstream). Por ejemplo:

* git push -u origin main

**Resumen:**

1. Crear nuestro directorio local.
2. Inicializamos nuestro repositorio.
3. Agregamos un control remoto en el servidor GitHub.

**En el desarrollo de trabajos:**

1. Agregamos nuestros archivos al area de preparación.
2. Hacemos uso de commit.
3. Empujamos nuestro desarrollo a la nube de GitHub.

**Actualización de repositorio: del servidor hacia el área de trabajo**

Para actualizar tu repositorio local con los cambios más recientes del repositorio remoto, puedes utilizar el comando:

* git pull

Este comando extrae los cambios del repositorio remoto y los fusiona con tu rama local.

**Git +**

**Eliminar un archivo de nuestro repositorio:**

* rm (nombre del archivo en el repo)
* git add (nombre del archivo en el repo)
* git commit -m "mensaje que ha eliminado un archivo, explica" -a

**Cambiar nombre de un archivo**

* git mv (nombre del archivo) (nombre del nuevo archivo)
* git add (nombre del archivo en el repo)
* git commit -m "mensaje que ha cambiado el nombre de un archivo, explica" -a

**Ver estado del Area de Preparación:**

* git status
* git status -s

**Crear un gitignore:**

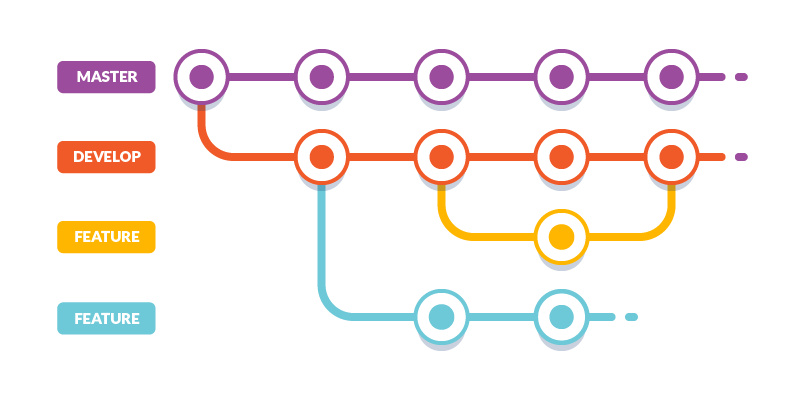
* touch .gitignore
* Agregar excepciones
* git add. gitignore
* git commit -m "mensaje"

**Hacer una comparación entre commits:**

* Mostrar identificadores: git log
* Mostrar identificadores, en resumen: git log --oneline
* Realizar la comparación: git diff (identificador1) (identificador2)

**Ramas en Git:**

En Git, las ramas son una característica fundamental que permite a los desarrolladores trabajar en diferentes versiones del proyecto de forma independiente. Cada rama representa una línea de desarrollo separada que puede tener su propio conjunto de cambios, historial de commits y archivos.



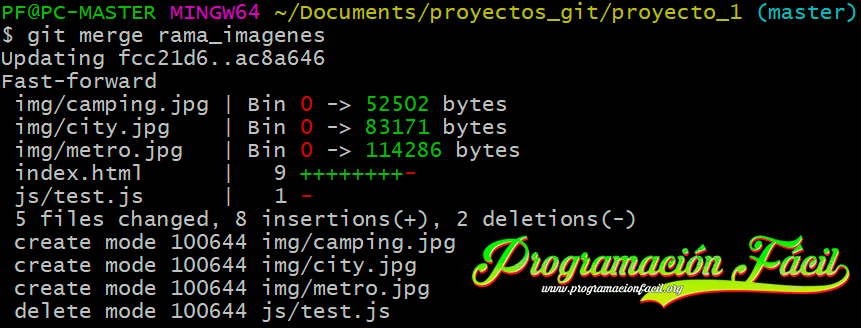
**Imagen con propiedad del autor ©MEDIUM**

**Funcionamiento de las ramas:**

Cuando creas un repositorio Git, se crea automáticamente una rama principal llamada "master" (o "main", dependiendo de la configuración). Esta rama es la línea principal de desarrollo y generalmente refleja la versión estable del proyecto.

Sin embargo, puedes crear nuevas ramas para trabajar en nuevas características, experimentar o solucionar problemas sin afectar la rama principal. Cuando creas una nueva rama, esta comienza como una copia exacta de la rama desde la cual se creó. A medida que realizas cambios y commits en una rama, esos cambios permanecen en esa rama hasta que decides fusionarlos con otra rama. Aquí comparto los comandos para la creación de ramas:

* Crear una rama: git branch (nombre de la rama)
* Trasladarse de una rama a otra: git switch (nombre de la rama)
* Fusionar una rama con la rama master: git merge (nombre de la rama)



**Imagen con propiedad del autor ©PROGRAMACION FÁCIL**

**GitHub:**

GitHub es una plataforma de desarrollo de software basada en la web que ofrece servicios de alojamiento de repositorios Git, herramientas de control de versiones y una variedad de características colaborativas para desarrolladores. Fue fundada en 2008 por Chris Wanstrath, PJ Hyett y Tom Preston-Werner, y desde entonces se ha convertido en una de las herramientas más populares para el desarrollo de software colaborativo.

**Características Principales:**

* **Repositorios Git:** GitHub proporciona alojamiento de repositorios Git, lo que permite a los desarrolladores almacenar, gestionar y colaborar en proyectos de software utilizando el sistema de control de versiones Git. Cada proyecto alojado en GitHub tiene su propio repositorio, que contiene el historial completo de cambios realizados en el código.
* **Control de Versiones:** GitHub facilita el seguimiento de las diferentes versiones de un proyecto de software a lo largo del tiempo. Los desarrolladores pueden realizar commits para guardar cambios en el repositorio, crear ramas para trabajar en características o correcciones de errores específicas, y fusionar ramas para combinar los cambios de forma segura.
* **Colaboración:** GitHub ofrece una variedad de características diseñadas para facilitar la colaboración entre desarrolladores. Esto incluye la capacidad de revisar y comentar el código, realizar solicitudes de extracción (pull requests) para proponer cambios en el código base, y discutir problemas y solicitudes de funciones en un tablero de problemas integrado.
* **Integraciones y Herramientas:** GitHub se integra con una amplia gama de herramientas y servicios, lo que facilita la integración continua, la entrega continua y otras prácticas de desarrollo ágil. También ofrece características como wikis, páginas de proyectos y un sistema de seguimiento de problemas para ayudar a los equipos a organizar y documentar su trabajo.
* **Impacto y Popularidad:** GitHub ha tenido un impacto significativo en la forma en que se desarrolla el software en la actualidad. Ha democratizado el proceso de desarrollo de software al proporcionar herramientas poderosas y accesibles para que los desarrolladores trabajen juntos de manera colaborativa, independientemente de su ubicación geográfica.

GitHub se ha convertido en la plataforma de elección para millones de desarrolladores y organizaciones de todo el mundo, incluidas algunas de las empresas más grandes y exitosas del sector tecnológico. Su popularidad se debe en parte a su facilidad de uso, su amplia gama de características y su sólida comunidad de desarrolladores.

**Elaborar la solución y explicación de la creación de un tablero M x N, asimismo, la actualización del tablero.cpp usando el primer apellido del estudiante o una "X" en la diagonal.**