



Tecnicatura Universitaria
en Programación

LABORATORIO DE COMPUTACIÓN III

Anexo N°1:
Versionador de código GIT

Material de Estudio
1° Año – 1° Cuatrimestre



Índice

GIT	2
Instalación de git	2
Elementos de git.....	3
Git Bash	4
Configuración	4
Get Started	6
GitHub	16
Creación de la cuenta	16
BIBLIOGRAFÍA	17

GIT

Git es un sistema de **control de versiones distribuido de código abierto**, que fue creado por Linus Torvalds en 2005 para el desarrollo del kernel de Linux. Git es ampliamente utilizado en la comunidad de desarrollo de software para controlar y gestionar el código fuente de los proyectos.

A diferencia de los sistemas de control de versiones centralizados, en los que todos los desarrolladores trabajan en un mismo repositorio central, Git permite que cada desarrollador tenga una copia completa del repositorio en su propio equipo. Esto hace que Git sea más resistente a los fallos y más rápido en las operaciones de clonación y fusión de ramas de desarrollo.

Git utiliza un modelo de ramificación flexible que permite a los desarrolladores crear ramas de desarrollo independientes para trabajar en nuevas características o arreglar errores, sin afectar la rama principal del proyecto. Además, proporciona una serie de herramientas para la resolución de conflictos y la colaboración entre desarrolladores.

Git se utiliza en una gran variedad de proyectos de software, desde pequeñas aplicaciones hasta proyectos empresariales de gran envergadura. También es compatible con una gran cantidad de plataformas de alojamiento de repositorios, como GitHub, GitLab o Bitbucket.

Instalación de git

Para instalar Git usaremos la información y las herramientas que se proveen en el sitio <https://git-scm.com/downloads>. Allí encontrarán como instalar git en cualquiera de los sistemas operativos mas comunes ([macOS](#), [Windows](#), [Linux/Unix](#))

Para los fines de esta materia, explicaremos como hacerlo en el sistema operativo Windows:

1. Acceder al siguiente link <https://git-scm.com/download/win>
2. De acuerdo a la versión de su sistema operativo, descargamos el ejecutable de git y lo ejecutamos como administradores.
3. Ya en la pantalla de instalación, Git posee una licencia GNU pública, aceptaremos los términos y condiciones y le damos al botón "Install"

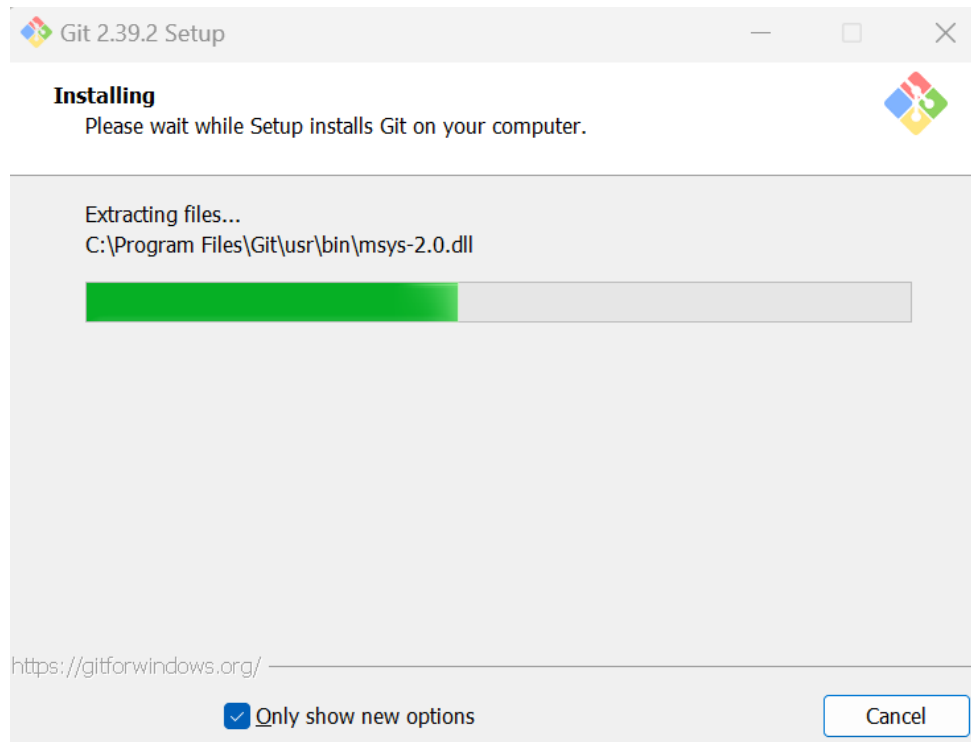


Imagen 1: Elaboración propia.

4. Una vez que termina la instalación, le damos al botón "Finish"

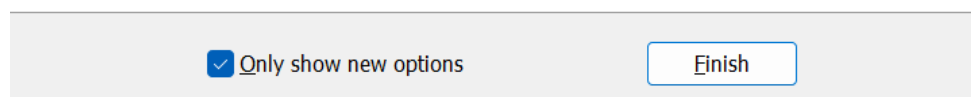


Imagen 2: Elaboración propia.

Elementos de git

La instalación de Git por defecto incluye varios componentes básicos que son necesarios para utilizar Git en la línea de comandos. Estos componentes pueden variar dependiendo del sistema operativo que se esté utilizando, pero en general, la instalación de Git incluye lo siguiente:

1. El sistema de control de versiones Git en sí mismo, que es el núcleo del software y permite la gestión de versiones de los archivos.
2. Una interfaz de línea de comandos para Git, que permite a los usuarios interactuar con Git utilizando comandos de la terminal (Git Bash).
3. Un cliente de interfaz gráfica de usuario (GUI) para Git, que proporciona una interfaz gráfica de usuario para interactuar con Git. Este cliente puede variar dependiendo del sistema operativo y de la distribución de Git que se esté utilizando.
4. Documentación y ayuda, que incluye manuales y guías de usuario para aprender a utilizar Git.

5. Herramientas auxiliares, como scripts de instalación y configuración, que facilitan la instalación y configuración de Git.

Es importante tener en cuenta que, aunque la instalación de Git por defecto incluye estos componentes básicos, es posible que se necesiten herramientas adicionales para una configuración más avanzada o específica de Git, como integración con herramientas de desarrollo, plataformas de alojamiento de código o sistemas de automatización de pruebas. En estos casos, puede ser necesario instalar y configurar herramientas adicionales para completar el proceso de configuración de Git.

Git Bash

GitBash es una herramienta que proporciona una terminal Bash en sistemas operativos Windows, y que incluye la distribución de Git para Windows. Bash es un intérprete de comandos de Unix que permite a los usuarios interactuar con el sistema operativo utilizando comandos en la línea de comandos.

Con esta herramienta, los usuarios de Windows pueden utilizar Git en la línea de comandos como si estuvieran utilizando un sistema Unix. GitBash proporciona una interfaz de línea de comandos similar a la de Unix, lo que significa que los comandos de Git se pueden utilizar de la misma manera que en sistemas Unix.

Además de Git, también se incluyen otras herramientas de línea de comandos de Unix, como grep, awk y sed. Esto hace que GitBash sea una herramienta útil para desarrolladores que trabajan en sistemas operativos Windows, pero que prefieren utilizar herramientas de línea de comandos de Unix.

Al utilizar GitBash, los desarrolladores pueden realizar tareas de control de versiones de Git en Windows utilizando la línea de comandos. Esto incluye la creación de repositorios, la realización de commits, la creación y gestión de ramas, y otras tareas de Git.

Configuración

Una vez que has instalado Git, hay algunas configuraciones que debes establecer antes de comenzar a trabajar con Git. A continuación, se muestran algunos de los pasos que puedes seguir para configurar Git después de instalarlo.

Configuración de Usuario

Estos comandos establecen el nombre y el correo electrónico del usuario en la configuración global de Git. Esta información se utiliza para etiquetar los commits que realizas, y **es importante asegurarse de que sea precisa**.

Para el uso de esta herramienta dentro del alcance de esta asignatura, se deberán configurar los datos del alumno, usando el número de legajo y el apellido y nombre separados por (_) (guión bajo) como nombre de usuario y el correo electrónico provisto por la facultad.

1. Abrimos la aplicación git bash para que nos muestre la terminal de git.
2. Configuramos el nombre y correo electrónico del usuario

```
git config --global user.name "100000_MORAIS"
```

```
git config --global user.email "100000@tecnicatura.frc.utn.edu.ar"
```

Configurar el editor de texto

Este comando establece el editor de texto que Git utiliza para editar mensajes de confirmación y otros archivos de texto. Si no se establece este valor, Git utilizará el editor predeterminado de tu sistema operativo.

En el siguiente link pueden encontrar los editores de texto recomendados por git y cómo configurarlos. Para esta asignatura se recomienda configurar IntelliJ una vez que haya sido instalado.

<https://docs.github.com/en/get-started/getting-started-with-git/associating-text-editors-with-git>.

```
git config --global core.editor "nombre-del-editor"
```

Configurar la herramienta de diferencias

Este comando establece la herramienta de diferencias que Git utilizará para resolver conflictos de fusión. Si no se establece este valor, Git utilizará su propia herramienta de diferencias por defecto. Para esta asignatura se recomienda configurar IntelliJ una vez que haya sido instalado.

```
git config --global merge.tool "herramienta-de-diferencias"
```


Get Started

Git es un sistema de **control de versiones distribuido**, lo que significa que **cada usuario tiene una copia completa del repositorio de Git en su computadora local**, en lugar de depender de un servidor centralizado para almacenar el código. Esto permite a los desarrolladores trabajar de forma aislada en sus propias ramas o características, sin afectar el trabajo de otros desarrolladores.

El funcionamiento básico de Git se basa en la creación y gestión de commits. **Un commit es una instantánea del repositorio en un momento dado**, que incluye cambios realizados en uno o varios archivos. Cuando se crea un commit, Git almacena una referencia al estado anterior del repositorio, lo que permite a los desarrolladores retroceder en el tiempo y ver los cambios realizados a lo largo del tiempo.

Los repositorios de Git **pueden tener varias ramas, que representan diferentes líneas de desarrollo**. Los desarrolladores pueden crear nuevas ramas para trabajar en nuevas características o para realizar experimentos sin afectar el código principal del repositorio. Una vez que las nuevas características están listas para ser integradas en el código principal, los desarrolladores pueden fusionar su rama con la rama principal del repositorio.

Git también proporciona herramientas para la resolución de **conflictos, que pueden surgir cuando dos desarrolladores realizan cambios en el mismo archivo al mismo tiempo**. Cuando se produce un conflicto, Git muestra las diferencias entre las dos versiones del archivo y permite al desarrollador elegir qué cambios deben conservarse.

Además de la gestión de versiones, Git también proporciona herramientas para el trabajo en equipo y la colaboración en proyectos. Los desarrolladores pueden compartir ramas y commits con otros miembros del equipo, y Git también proporciona integración con plataformas de alojamiento de código como GitHub, que facilitan la colaboración y la gestión de proyectos a nivel de equipo.

Repositorios (repository)

Un repositorio en Git es un espacio de almacenamiento de versiones de archivos y carpetas. Puedes crear un repositorio desde cero o clonar un repositorio existente. Por ejemplo, para crear un nuevo repositorio en tu máquina, puedes utilizar el comando:

```
git init
```

Además de crear un repositorio, también se puede trabajar sobre un repositorio existente, haciendo una clonación del mismo. Clonar un repositorio es el proceso de crear una copia local de un repositorio remoto en tu máquina local. Al clonar un repositorio, se descarga toda la información del repositorio, incluyendo el historial de commits, las ramas y las etiquetas, y se crea una copia exacta del repositorio en tu máquina local. Esto permite trabajar en el repositorio localmente, hacer cambios y subir esos cambios de vuelta al repositorio remoto.

Para clonar un repositorio utilizando Git, se utiliza el comando `git clone` seguido de la URL del repositorio que se desea clonar. Por ejemplo:

```
git clone https://github.com/usuario/my-repo.git
```

Ramas (Branchs)

Las ramas en Git representan diferentes líneas de desarrollo. Puedes crear una nueva rama para trabajar en nuevas características o para realizar experimentos sin afectar el código principal del repositorio. Por ejemplo, para crear una nueva rama en Git, puedes utilizar el comando:

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

Stages (Add & Reset)

En Git, el área de preparación (también conocida como "stage" o "index") es un lugar intermedio entre el directorio de trabajo y el repositorio. Los archivos que se añaden al área de preparación se preparan para ser confirmados (commit) en el repositorio.

Para agregar archivos al área de preparación, se puede utilizar el comando `git add`. Por ejemplo, si se desea añadir el archivo **nuevo_archivo.txt** al área de preparación, se debe ejecutar el siguiente comando:

```
git add nuevo\_archivo.txt
```

Después de ejecutar este comando, Git añadirá el archivo **nuevo_archivo.txt** al área de preparación. El archivo estará listo para ser confirmado en el repositorio la próxima vez que se ejecute el comando `git commit`.

Para quitar archivos del área de preparación, se puede utilizar el comando `git reset`. Por ejemplo, si se desea quitar el archivo **nuevo_archivo.txt** del área de preparación, se debe ejecutar el siguiente comando:

```
git reset nuevo\_archivo.txt
```


Después de ejecutar este comando, Git quitará el archivo **nuevo_archivo.txt** del área de preparación. El archivo seguirá siendo modificado en el directorio de trabajo, pero ya no estará listo para ser confirmado en el repositorio hasta que se añada de nuevo al área de preparación con el comando git add.

Commits

Un commit en Git representa una instantánea del estado del repositorio en un momento dado. Un commit registra los cambios realizados en uno o varios archivos. Por ejemplo, para realizar un commit en Git, debes agregar los cambios a la zona de preparación (o staging area) utilizando el comando git add, y luego realizar el commit utilizando el comando git commit. Por ejemplo:

```
git add archivo1.txt archivo2.txt  
git commit -m "Agregando archivo1 y archivo2"
```

Push

El comando "git push" es utilizado para subir los cambios locales que se han realizado en un repositorio de Git a un repositorio remoto en la nube, como GitHub o GitLab. Al realizar un push, los cambios locales que se han realizado se envían al repositorio remoto para que otros colaboradores puedan acceder a ellos y trabajar en ellos.

Para utilizar el comando git push, primero es necesario haber realizado cambios locales en el repositorio. Luego, se debe utilizar el comando git add para agregar los cambios al área de preparación, y luego el comando git commit para confirmar los cambios y crear un nuevo commit. Una vez que se ha creado un commit con los cambios, se puede utilizar el comando git push para enviar los cambios al repositorio remoto. La sintaxis básica del comando git push es la siguiente:

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

Donde **<nombre-remoto>** es el nombre del repositorio remoto al que se está enviando los cambios, y **<nombre-rama>** es el nombre de la rama que se está subiendo. Por ejemplo, para enviar los cambios de la rama "main" al repositorio remoto "origin", se puede utilizar el siguiente comando:

```
git push origin main
```

Es importante recordar que se debe tener permisos de escritura en el repositorio remoto para poder realizar un push. Si no se tiene permisos de escritura, se puede hacer un fork del repositorio y realizar cambios en la copia del repositorio y enviar un pull request para solicitar que los cambios se integren en el repositorio original.

Status

El comando “git status” es una herramienta importante en Git que muestra el estado actual de los archivos en un repositorio. Es una forma rápida de verificar qué archivos han sido modificados, cuáles están en el área de preparación, y cuáles no están siendo rastreados por Git.

Cuando se utiliza el comando git status, se muestra una lista de archivos y su estado actual en el repositorio. Estos estados incluyen:

- Archivos **Changed**: Aquellos que se editaron desde la última confirmación.
- Archivos **Staged**: Aquellos que se han añadido al stage.
- Archivos **Untracked**: son aquellos que Git no está rastreando actualmente.

Supongamos que se ha creado un nuevo archivo llamado **nuevo_archivo.txt** en un repositorio Git. Si se utiliza el comando git status, se verá algo como esto:

```
git status
```

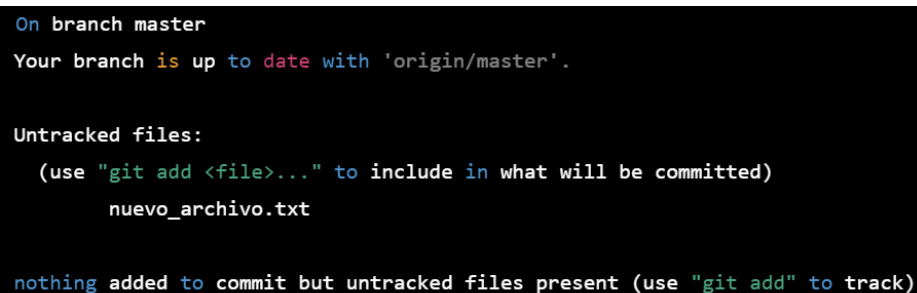
A terminal window with a dark background and light-colored text. The output of the 'git status' command is displayed. It shows the current branch is 'master' and it is up to date with 'origin/master'. It then lists 'Untracked files:' followed by 'nuevo_archivo.txt'. A note in parentheses suggests using 'git add' to include the file in the next commit. The final line states that nothing was added to the commit but that untracked files are present and can be tracked with 'git add'.

Imagen 3: Elaboración propia.

Este resultado indica que el repositorio está en la rama master y que está actualizada con la rama master en el repositorio remoto origin. También muestra que hay un archivo llamado **nuevo_archivo.txt** que no está siendo rastreado por Git. En otras palabras, Git no está al tanto de los cambios que se hagan en este archivo.

El mensaje "**Untracked files**" indica que se puede utilizar el comando git add para añadir el archivo al área de preparación de Git. De esta forma, Git comenzará a rastrear los cambios que se realicen en este archivo. Después de añadir el archivo al área de preparación, se podrá utilizar el comando git status de nuevo para verificar que el archivo ha sido añadido al área de preparación y está listo para ser confirmado en el repositorio.

Ahora supongamos que se ha modificado el archivo **nuevo_archivo.txt** y se ha añadido una nueva línea de texto. Si se utiliza el comando git status, se verá algo como esto:

```
git status
```

```
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
       modified:   nuevo_archivo.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Imagen 4: Elaboración propia.

Este resultado indica que el archivo **nuevo_archivo.txt** ha sido modificado pero no se ha añadido al área de preparación (stage). El mensaje "**Changes not staged for commit**" indica que se debe utilizar el comando git add para añadir el archivo al área de preparación y prepararlo para ser confirmado en el repositorio.

Después de añadir el archivo al área de preparación, se puede utilizar el comando git status de nuevo para verificar que el archivo ha sido añadido al área de preparación y está listo para ser confirmado. El resultado:

```
git status
```

```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
       modified:   nuevo_archivo.txt
```

Imagen 5: Elaboración propia.

Este resultado indica que el archivo **nuevo_archivo.txt** ha sido añadido al área de preparación y está listo para ser confirmado en el repositorio. Se puede

utilizar el comando git commit para confirmar los cambios y añadirlos al historial de confirmaciones del repositorio.

History (Log)

El historial de Git (también conocido como "Git log") es una herramienta que permite ver la lista de confirmaciones (commits) que se han realizado en un repositorio. El historial muestra la información relacionada con cada confirmación, como el identificador de confirmación (hash), el autor, la fecha, el mensaje de confirmación y los cambios realizados.

El comando git log se utiliza para ver el historial de confirmaciones. Por ejemplo, si se desea ver el historial de confirmaciones de la rama actual, se debe ejecutar el siguiente comando:

```
git log
```

```
commit a7f97c80f5eb6e5b6d2d6c5b24de4f5294a9a4a8
Author: John Doe <johndoe@example.com>
Date:   Fri Feb 18 10:36:25 2022 -0500

    Add new feature

commit 7d3e72c3f8d3c6feecf7cdefbc9cb9b8d173a95a
Author: Jane Smith <janesmith@example.com>
Date:   Wed Feb 16 09:22:13 2022 -0500

    Fix bug in login form

commit 3c4678e3b3c3b69b2d1f2a8ca3f6a3c7dcdb23e2
Author: John Doe <johndoe@example.com>
Date:   Mon Feb 14 16:45:29 2022 -0500

    Update homepage layout

...
```

Imagen 6: Elaboración propia.

Este comando mostrará una lista de confirmaciones en orden cronológico inverso (la confirmación más reciente aparece primero), junto con la información relacionada con cada confirmación.

Fusiones (Merge)

Las fusiones en Git permiten combinar dos ramas en un único conjunto de cambios. Puedes fusionar una rama en otra rama utilizando el comando `git merge`. Por ejemplo, si tienes una rama llamada "feature-branch" que quieres fusionar en la rama principal "master", puedes utilizar el siguiente comando:

```
git checkout master
```

```
git merge feature-branch
```

Fork

Un fork en Git es una copia de un repositorio existente en una cuenta de usuario diferente. El proceso de fork permite a los usuarios realizar cambios en un repositorio sin afectar directamente al repositorio original. En otras palabras, un fork es una bifurcación o ramificación del repositorio original en el que se crea una copia independiente del proyecto.

El proceso de realizar un fork no se realiza mediante un comando de Git, sino que se realiza en la plataforma de alojamiento del repositorio.

El proceso de realizar un fork es simple y se realiza en la plataforma de alojamiento del repositorio. Por ejemplo, en GitHub se puede realizar un fork de un repositorio utilizando el botón "Fork" en la página del repositorio. Al hacer un fork, se crea una copia del repositorio en la cuenta del usuario, y se pueden hacer cambios en la copia sin afectar el repositorio original. Una vez que se han realizado los cambios, se puede crear un pull request para solicitar que los cambios se fusionen en el repositorio original.

Los forks son muy útiles en proyectos de software de código abierto y colaborativos, ya que permiten que los usuarios realicen cambios en el código sin afectar directamente al repositorio original. Esto permite que los colaboradores experimenten con nuevas ideas y funciones sin afectar a la estabilidad del proyecto principal. Además, los forks también permiten que se realice un seguimiento de diferentes versiones del mismo proyecto, lo que puede ser útil para proyectos que tienen diferentes variantes o versiones.

Pull request (PR)

Un pull request (PR) es una solicitud de colaboración en Git, específicamente en plataformas de alojamiento de repositorios como GitHub, GitLab o Bitbucket. Un pull request es una forma de proponer cambios en un repositorio remoto y solicitar a

otros colaboradores que revisen y aprueben esos cambios antes de incorporarlos en la rama principal del repositorio.

El proceso de crear un pull request generalmente implica los siguientes pasos:

1. Crear una rama de trabajo a partir de otra rama del repositorio.
2. Realizar cambios en la nueva rama de trabajo, ya sea modificando archivos existentes o agregando nuevos archivos.
3. Hacer un commit con los cambios y enviarlos a la rama de trabajo.
4. Abrir un pull request en el repositorio remoto y solicitar a otros colaboradores que revisen y aprueben los cambios propuestos.
5. Si se detectan problemas o se sugieren mejoras, el autor del pull request puede realizar cambios adicionales en la rama de trabajo y actualizar el pull request.
6. Una vez que el pull request es aprobado, los cambios se incorporan en la rama principal del repositorio y se cierra el pull request.

El proceso de pull request es útil para los proyectos de software de código abierto y colaborativos, ya que permite a los colaboradores revisar y comentar los cambios propuestos antes de incorporarlos en la rama principal. También ayuda a mantener un historial claro y completo de los cambios realizados en el repositorio, lo que facilita la detección de errores o la reversión de cambios problemáticos.

Resolución de conflictos

En Git, **los conflictos ocurren cuando dos o más personas hacen cambios en el mismo archivo al mismo tiempo y esos cambios entran en conflicto**. En tales casos, Git no sabe qué cambios tomar, y es necesario que un desarrollador resuelva el conflicto manualmente. La resolución de conflictos es una parte importante de trabajar con un sistema de control de versiones distribuido como Git.

1. Para resolver un conflicto en Git, se pueden seguir los siguientes pasos:
2. Comprobar el estado del repositorio: primero, se debe comprobar el estado del repositorio utilizando el comando `git status`. Este comando mostrará los archivos que tienen conflictos y la rama en la que se produjo el conflicto.
3. Abrir el archivo con conflictos: el siguiente paso es abrir el archivo con conflictos en un editor de texto y buscar las secciones que generaron conflictos. Las secciones en conflicto estarán entre líneas de `<<<<<<`, `=====`, y `>>>>>>`.
4. Editar el archivo para resolver los conflictos: para resolver los conflictos, se deben editar manualmente las secciones en conflicto y tomar las decisiones

necesarias para elegir la versión correcta del código. Por ejemplo, se puede decidir utilizar las líneas de código que aparecen en la rama actual, la rama remota o ambas versiones.

5. Añadir el archivo y confirmar los cambios: una vez que se han resuelto los conflictos en el archivo, se debe utilizar el comando `git add` para añadir el archivo al área de preparación y luego el comando `git commit` para confirmar los cambios.
6. Hacer un push de los cambios: después de haber confirmado los cambios, se debe hacer un push de los cambios al repositorio remoto utilizando el comando `git push`.

Aquí te muestro un ejemplo de cómo se vería un conflicto en Git:

Supongamos que dos desarrolladores han trabajado en el mismo archivo y han realizado cambios diferentes en las mismas líneas. Después de intentar fusionar ambas ramas, Git muestra el siguiente mensaje:

```
Auto-merging myfile.txt
CONFLICT (content): Merge conflict in myfile.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Imagen 7: Elaboración propia.

En este caso, Git detectó que el archivo "**myfile.txt**" tiene un conflicto de fusión y no pudo realizar la fusión de manera automática. El mensaje indica que el conflicto está relacionado con el contenido del archivo.

Para resolver el conflicto, se debe abrir el archivo "**myfile.txt**" y encontrar las secciones que Git ha marcado como conflictivas. Estas secciones se verán como algo similar a esto:

```
<<<<<< HEAD
Línea(s) modificada(s) en rama actual
=====
Línea(s) modificada(s) en la otra rama
>>>>>> other-branch
```

Imagen 8: Elaboración propia.

En este ejemplo, la sección <<<<<< **HEAD** indica que las líneas debajo de ella son las que se encuentran en la rama actual, mientras que >>>>>> **other-branch** indica que las líneas debajo de ella son las que se encuentran en la otra rama.

Para resolver el conflicto, se debe editar el archivo para eliminar las marcas de conflicto y decidir qué cambios se deben mantener y cuáles descartar. Después de hacer los cambios necesarios, se debe guardar el archivo y hacer un nuevo commit para completar la fusión.

Todos lo comandos

Estos conceptos y comandos explicados aquí arriba pueden considerarse los principales de git y probablemente sean los que más se usen durante todo el ciclo de desarrollo del software. Para conocer y aprender sobre todos los comandos de git, se pueden encontrar en el sitio oficial de la herramienta <https://git-scm.com/docs>

GitHub

GitHub es una plataforma de alojamiento de repositorios de Git basada en la web. Fue adquirida por Microsoft en 2018 y es una de las plataformas de alojamiento de código más utilizadas en la comunidad de desarrollo de software. Ofrece un amplio conjunto de características para la gestión de proyectos de software. Algunas de sus principales características incluyen:

- Repositorios públicos y privados: Los repositorios pueden ser públicos (accesibles para todo el mundo) o privados (accesibles solo para los colaboradores autorizados).
- Control de versiones: GitHub ofrece una gestión completa de las versiones del código fuente, permitiendo la creación y fusión de ramas de desarrollo, y facilitando la resolución de conflictos.
- Colaboración: GitHub permite a los colaboradores de un proyecto trabajar juntos de manera eficiente, mediante el uso de solicitudes de extracción (pull requests), discusiones, revisión de código y comentarios.
- Integraciones: GitHub es compatible con una amplia variedad de herramientas y servicios, como integración continua, pruebas automatizadas, despliegue continuo, entre otras.
- Seguridad: GitHub ofrece una serie de características para la seguridad del código fuente, como autenticación de dos factores, control de acceso basado en roles y revisiones de seguridad automatizadas.

Además de estas características, GitHub es conocida por su gran comunidad de desarrolladores y su amplia variedad de proyectos de código abierto disponibles para su exploración y contribución. También es una plataforma popular para alojar proyectos de software de código abierto.

Creación de la cuenta

Para el desarrollo de esta asignatura, usaremos la plataforma de gitHub para alojar todos los repositorios de ejemplos de código, ejercicios y exámenes. Para poder hacer uso de la misma, deberán crear un usuario en la plataforma **usando el email institucional**. En el siguiente link está el detalle de como crear la cuenta en GitHub: <https://github.com/signup>

BIBLIOGRAFÍA

Página oficial de GIT(<https://git-scm.com/>).

Página oficial de GitHub (<https://github.com/>).



Atribución-No Comercial-Sin Derivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterado su contenido, ni se comercialice. Universidad Tecnológica Nacional Facultad Regional Córdoba (S/D). Material para la Tecnicatura Universitaria en Programación, modalidad virtual, Córdoba, Argentina.