Programación I

> 16 de abril de 2025, clase 01 de Programación I

¿Qué es un IDE?

Un **IDE** es un Entorno de programación estructurad que se compone de:

1. **Lenguaje de Programación**: *(Debe estar instalado en tu pc)*.
2. **Compilador del Lenguaje a utilizar**: *Un programa que se encargará de ejecutar el código oque tu escribirás*.
3. **Editor de texto**: *Donde tu escribirás tu código*, es recomendable utilizar Visual studio code junto con sus extensiones para tener una mejor experiencia.
4. **Terminal de mi Lenguaje**: *Una ventana en la yo pueda ejecutar los comandos relacionados al lenguaje que voy a trabajar*.
5. **Git-Bash**: *Una herramienta para integrar tu proyecto en la red*.

Algunos comandos clave para el terminal

> 21 de abril de 2025, clase 02 de Programación I

Algunos atajos de teclado para el Visual code

- *F1 o CTRL+ SHIFT+ P*: abre una barra de busqueda de comandos relacionados a las extenciones del visual, **ejemplo**: Abrir un preview o generar un pdf.
- *CTRL+ B*: Esconde la barra lateral que muestra a los archivos y/o workspaces abiertos.
- *AlT+ up,down,left,right*: me permite mover las líneas de mi código.
- *SHIFT+ ALT+ A*: me permite generar una multilinea.
- *CTRL+ SPACE*: activar sugerencias (para tomarlas usar **TAB**).
- *SHIFT+ ALT+ UP/DOWN*: copiar una línea de código.
- *CTRL+ P*: abrir rápidamente un proyecto.
- *CTRL+ D*: cursor múltiple.
- *CTRL+ T*: buscar símbolo o palabra que que hayas seleccionado.
- *CRTL+ K+ C*: el código que selecciones se convertirá en comentario
- *CTRL+ K+ U*: las líneas de comentarios seleccionadas dejarán de ser comentario.

Comandos para el CMD o Powershell (win y Linux)

- **pdw**: muestra a que directorio está asociada la terminal
- **ls**: muestra los archivos de un directorio.
- **cd..**: la terminal se mueve un directorio más arriba.
- **touch *archivo.extensión*** , o en su defecto **touch directorio/*archivo.extensión*** para crear archivos.
- **ls** o **cat** o **code *archivo.extensión*** *(si hace falta, poner directorio antes del archivo)* para leer un archivo.
- **mkdir**: crear un directorio o carpeta.
- **rm *archivo.extensión*** *(agregue directorio antes del archivo si es necesario)* para eliminar un archivo.
- **cp *archivo(a copiar).extensión directorio/nombre de la copia.extensión*** para copiar un archivo dado.

- **mv *archivo.extensión directorio donde quieras mandarle/*** para mover un archivo de lugar o renombrarlo.
- **ps**: procesos activos
- **ipconfig**: para ver direcciones ip (incluso puedes ver la de tu router).
- **ping *web.com***: analiza la conexión con un sitio web.
- **echo "hola mundo">>*archivo.extensión***: inprime un mensaje en un arhivo dado.

Algunos Comandos exclusivos del GIT

Nótese que siempre empiezan con "git"

-**git config --global -h**: ayuda para configurar cosas del GIT

- **git config --global user.name** / **email** para consultar con que credenciales el equipo esta registrado en la red del GIT. agregar "*usuario / email que quieras configurar*" despues del comando de arriba para configurar usuario y email.

Ojo! Debes configurar usuario y email para poder subir tu proyecto a la nube

- **git init**: iniciar el monitoreo de GIT (puntos de control) en el workspace en el que estás.
- **git add *archivo.extensión***: agregar un archivo al monitoreo de GIT. **git add .** agregará todos los archivos de un directorio al trakeo de GIT.
- **git status**: reporte de la situación por parte de GIT (lo que se modificó o sincronizó).
- **git commit -m "*mensaje*"**: sincronizar cambios detectados por el GIT en los documentos que monitorea.
- **git++ *archivo.cpp -o ejecutable.exe***: para compilar un archivo de código de C.
- **directorio/archivo.exe**: para ejecutar un archivo .exe desde la consola.

> 22 de abril de 2025, clase 03 de Programación

GIT para la nube

- *git push*: para subir tu proyecto a un repositorio en la nube.
- *git pull*: para descargar código de un repositorio de la nube.
- *git clone URL.com*: para clonar un repositorio de la web.

Algunas carpetas importantes en tu Workspace

- **Bin**: binaries, los .exe generados despues de copilar código.
- **Lib**: librerías o bibliotecas a utilizar en el desarrollo.
- **Src**: donde se alojará el código base del proyecto
- **Database**: donde se encontrarán todos los archivos relacionados a los datos.
- **Tmp**: archivos temporales o auxiliares, puede alojar un archivo *gitignore*
- Si:
  touch directorio/.gitignore

  echo "<>.formato">>.gitignore*

> Imprimiré un parametro en *gitignore* para hacer que git no trakee a los archivos de cierto formto, esto es util con archivos .exe,.pdf,.htlm, ya que se generan cada que se compile un código.

mi-proyecto/

├── .env

├── bin/

│ └── setup.sh

├── lib/

│ └── utils.js

├── database/

│ ├── schema.sql

│ └── seed_data.sql

├── src/

│ ├── main.js

│ └── controllers/

│ └── userController.js

├── temp/

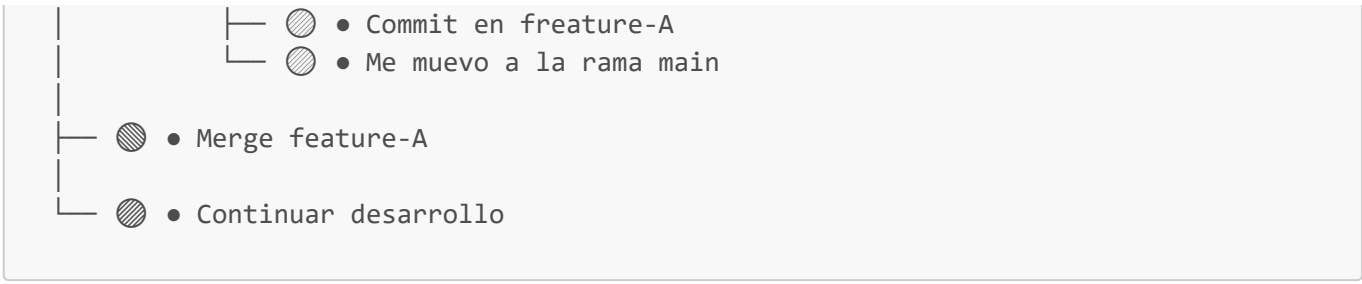
│ └── cache/

└── README.md

Branchs de GIT en tu Proyecto

Los diferentes puntos de control que crea GIT en tu proyecto se alojan a lo largo de una rama principal (tienen un id y hash asociados), aunque puedes crear más ramas paralelas que estrán dedicadas a desarrollar diferentes partes del proyecto. Toma en cuenta que al final del desarrollo las ramas se tendrán que unir de nuevo a la principal mediante una función "merge".

```
main
|
├── ●Commit I
|
├── ●Commit II
|
├── ●Commit III
|     |
|     └── feature/A
|           |
|           ├── ● Desarrollo feature-A
```



> 23 de arbril de 2025, clase 04 de Programación I

Comandos Importantes para las Branchs de GIT

- *git branch*: muestra todas las ramas creadas
- *git branch -m nombre_nuevo*: renombrar una branch
- *git log*: listado de todos los puntos de control de tu branch

  git log --graph: agrega un pequeño dibujo al comando de arriba, recuerda salir del log con *q* (quit).
- **git checkout archivo.formato**: vuelve al último punto de control que haya guardado GIT.

  git checkout *hash del punto de guardado*: para volver a un punto de control específico.
- **git tag *nombre_custom***: darle un nombre personalizados a tus commits o puntos de control (ver1 ,ver2, ver3, etc).
- *git diff*: reporta los cambios de código dados despues de un commit
- *git reset*:

C/C++

> 28 de abril de 2025, clase 05 de Programación I

Manejo de Branchs en GIT

Pueden existir tantas branch como funcionalidades tenga el proyecto, algunas se encargarán de resolver problemas específicos como la aparición de errores críticos por medio de un "hotfix".

- **git branch *nombre***: crea y nombra un branch a partir de la rama principal. *la rama será creada a partir del último commit guardado en la rama madre*.
- **git switch *nombre_rama_objetivo***:cambiar la rama en la que te encuentras por la que indicas en el comando.
- **git merge *rama_externa***: Une los elementos que tenga la rama mencionada (a partir de su último commit) en el comando con los elementos que tenga la rama en la que se encuentere en ese momento.

Unificar dos Ramas

Se hace un commit de la rama en la que se este trabajando, para luego cambiarse a la rama principal desde la cual se ha de ejecutar la función *merge*.

El proceso completo de drear una rama nueva, trabajar en ella y luego unir los cambios a la rama principal queda tal que:

-git branch freatures

-git switch freatures

-git commit -m ""

-git switch main

-git merge fratures

-git checkout -b freature (**opcional**: esta parte borra la rama mencionada en el comando)