



UNIVERSIDAD DE CÓRDOBA

Departamento de Informática y Análisis Numérico

**Ingeniería del Software, Conocimiento y
Bases de Datos**

Campus Universitario de
Rabanales
Edificio Albert Einstein,
Planta 3
14071 Córdoba (ESPAÑA)



Ingenieria de Software

Author : Adil, Hirchi
profesora : Aurora Ramirez Quesada

Practica 1: Git

Córdoba, septiembre ,2020



INDICE DE CONTENIDOS

1.Propósito del documento.....	
2.¿Que es un sistema de control de versiones?.....	
2.1.sistema de control de versiones centralizado & distribuido.....	
2.2.Git frente a otros sistema de control de versiones.....	
3.configuraicon de Git “establecer nombre y email”	
4.crear nuevo repositorio	
5.crear fichero en el repositorio.....	
6.añadir ficheros en el repositorio.....	
7.compilar y comprobar el estado de git.....	
8.crear rama brnachA.....	
9.confirnar cambios.....	
10.apuntar a la rama master y modificar nuestros ficheros.....	
11.fusionar dos ramas	
12.crear conflicto entre dos ramas.....	
13.subir Repositorio a github.....	
14.bajar repositor desde github a nuestro repositorio local git.....	



1. Propósito del documento :

En este documento vamos a ver uno de los sistemas de gestión y control de versiones más importantes llamado “Git”, desarrollando una serie de directivas para transmitir unos conceptos básicos de gestión y control de nuestros proyectos por medio de la herramienta Git.

Desde un software libre “el kernel de linux” de código abierto nace otro software libre Git, ya que gracias al creador de linux “linus torvalds” y al desarrollo del proyecto de linux que enfrente al fracaso del sistema de gestión de proyectos llamado Bitkeeper que usaban en el desarrollo del proyecto linux, condujo a linus torvald a pensar en crear un sistema propio para la gestión y control de su proyecto y por fin da luz al sistema de gestión y control de versiones Git.

2. ¿Que es un sistema de control de Versiones ? :

En lo habitual el desarrollo de un proyecto software se centraban mas en desarrollo del proyecto que en la gestión y control del proyecto . De una manera mecánica copiando archivos y códigos de una carpeta a otra y agrupando trabajos entre varias carpetas navegando en nuestro propio directorio personal . Lo que conlleva a una serie de complejidad muy alta y costoso en la gestión y control de nuestras versiones , la localización y el seguimiento del desarrollo de nuestro proyecto en grupo .



Entonces surgen sistema de control de versiones que nos permiten la gestion y control de versiones de nuestro proyecto de una manera sencilla,simples y facil con una interfaz de funciones y operaciones bastantes intuitivas y basicas para dominar de forma eficiente y eficaz nuestro desarrollo del proyecto y sus actualizaciones . Permittiendonos navegar de un proyecto a otro con un simple paso con alta velocidad y rendimiento .

2.1. Sistema de control de versiones distribuido & centralizado :

tal y como mencionamos en el apartado anterior,un sistema de control de versiones vino con el fin de solucionar el problema de gestion y siguimiento de nuestro proyecto pero es suficiente para poder controlar nuestras versiones en nuestro computador personal? ...efectivamente no,ya que el desarrollo de un proyecto software mayormente suele desarrollarse en entorno grupal ,por lo tanto tener nuestro proyecto en un sistema personal,no resuelve problema de control y siguimiento del proyecto .



**Para solucionar el problema de trabajo en grupo .
Surge el concepto de servidor central o sistema de
control de versiones centralizado que nos permitira a
parte de control de las versiones de nuestro proyecto
tambien poder compartir nuestro proyecto y su
siguimiento en grupo .**

**Permitiendo asi a nuestros miembros del grupo
participar en el trabajo y su acceso completo al
proyecto .**

**Con este sistema de control de versiones
centralizado queda resuleto el problema de compartir
nuestro proyecto entre grupos.pero que pasara a
nivel de seguridad de nuestra informacion? Cuantas
transicciones son necesarias cada instantes o
actualizacion de nuestro trabajo? Es responde a la
integridad y consistencia de nuestro sistema frente
cualquier fallo de la red ? ..etc para resolver todas
esa cuestiones y problemas que puede sufrir un
sistema de control de versiones centralizado surgen
las sistema de control Distribuidos que nos permite a
parte de compartir el proyecto en grupo y su
siguimiento,nos permite tener respuestas
instantaneas y acceso rapido con alto rendimiento**



creando replicas de todo nuestro proyecto en el disco duro local de

cada miembro del grupo dando una respuesta rapida y la posibilidad de trabajar desde cualquier punto sin necesidad de acceso instantaneo al servidor central .

2.2.Git frente a otras sistemas de control de versiones (mercurial..etc):

Entre las características mencionadas anteriormente de Git como sistema de gestion y control de versiones solucionando el problema de distribucion del trabajo,el rendimiento,la disponibilidad . Git proporciona otro característica muy notable enfrente otros sistemas de Cvs que reside en la integridad de la informacion y sus versiones .

Ya que Git es muy sencible a las modificaciones que viven nuestros archivos en esepecifico y nuestros proyectos a nivel general permitiendo crear y almacenar instantaneas para cada modificacion generando una nueva version de todo nuestro proyecto y no solo almacena el cambio particular



ocurrido en un cierto fichero .lo que difirencia
mayormente Git de las demas sistema de Cv.

Ofreciendo un sistema de seguridad muy alto
encriptando la informacion de forma rapida
instantanea como una serie de bits o informacion
encriptada por la funcion Sha .

3.Configuracion de Git “establecer nombre y email” :

```
adil@adil-HP-Laptop-15s-fq1xxx: ~/git
dil@adil-HP-Laptop-15s-fq1xxx:~/git$ git config --global user.name "Adil"
dil@adil-HP-Laptop-15s-fq1xxx:~/git$ git config --global user.name
dil
dil@adil-HP-Laptop-15s-fq1xxx:~/git$ git config --global user.email "adil.infosoft@gmail.com"
dil@adil-HP-Laptop-15s-fq1xxx:~/git$ git config --global user.email
dil.infosoft@gmail.com
dil@adil-HP-Laptop-15s-fq1xxx:~/git$
```

4. crear Nuevo repositorio :

para crear un nuevo repositorio usamos el comando
“git init” ,que crea un nuevo subdirectorio
llamado .git donde se almacenan todos los archivos y
lo que vamos manipulando durante el desarrollo de
nuestro proyecto .



```
adil@adil-HP-Laptop-15s-fq1xxx: ~/git/proyect-git-is
adil@adil-HP-Laptop-15s-fq1xxx:~/git$ mkdir proyect-git-is
adil@adil-HP-Laptop-15s-fq1xxx:~/git$ cd proyect-git-is
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is$ git init
initialized empty Git repository in /home/adil/git/proyect-git-is/.git/
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is$
```

Cuando se crea el nuevo repositorio .automaticamente git genera el nuevo subdirectorio .git que contien los siguientes ficheros :

```
adil@adil-HP-Laptop-15s-fq1xxx:~$ cd /home/adil/git/proyect-git-is/.git/
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/.git$ ls
branches  config  description  HEAD  hooks  info  objects  refs
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/.git$
```

en la siguiente tabla realizamso una breve descripcion de cada archivo componente del subdirectorio .git



Fichero	Descripcion
Config	Contiene las opciones de configuraciones de nuestro proyecto
info	Almacena un archivo con los patrones principales a ignorar
hooks	Contiene nuestros scripts
Objects	Guarda contenido de nuestro base de datos
Refs	Guarda los apuntadores de referencia a nuestros commit confirmados
head	Apuntador dinamico que apunta a la rama que tengamos activa
Index	Guarda toda la informacion relacionada a nuestra area de archivos en estado de preparacion y listos para ser confirmados



5. crear directorio "include" :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is$ mkdir include
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is$ cd include
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

6. crear ficheros en el repositorio :

6.1. crear fichero helloworld.c :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ gedit helloworld.c &
[1] 7680
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

6.2. escribir el código :

```
Open  [icon] *helloworld.c
~/gitslycs/include

#include "./include/myinclude.h"
int main(){
    f();
    return 0;
}
```

6.3. crear el fichero myinclude.h :

```
[1] 14626
adil@adil-HP-Laptop-15s-fq1xxx:~/gitslycs/include$ gedit myinclude.h &
[2] 14858
adil@adil-HP-Laptop-15s-fq1xxx:~/gitslycs/include$
```



6.4.codigo myinclude.h :

```
*helloworld.c  x  myinclude.h
1 #include <stdio.h>
2 void f()
3 {
4 printf("Hello World \n");
5
6 }
```

6.5.comprobar estado del repositorio :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  ./

nothing added to commit but untracked files present (use "git add" to track)
[2]+  Done                  gedit helloworld.h
```

6.6.agregar los cambios al stage :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git add .
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   helloworld.c
    new file:   helloworld.h

adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```



6.7.confirmar los cambios :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "initial commit"
[master (root-commit) 1bbd543] initial commit
2 files changed, 18 insertions(+)
create mode 100644 include/helloworld.c
create mode 100644 include/helloworld.h
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

6.8.comprobar estado :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git status
On branch master
nothing to commit, working tree clean
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

7.compilar nuestro codigo :

```
adil@adil-HP-Laptop-15s-fq1xxx: ~/git/proyect-git-is/include
[2]+  Done                  gedit myinclude.h
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ gcc helloworld.c -o helloworld
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ ./helloworld
hello world
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

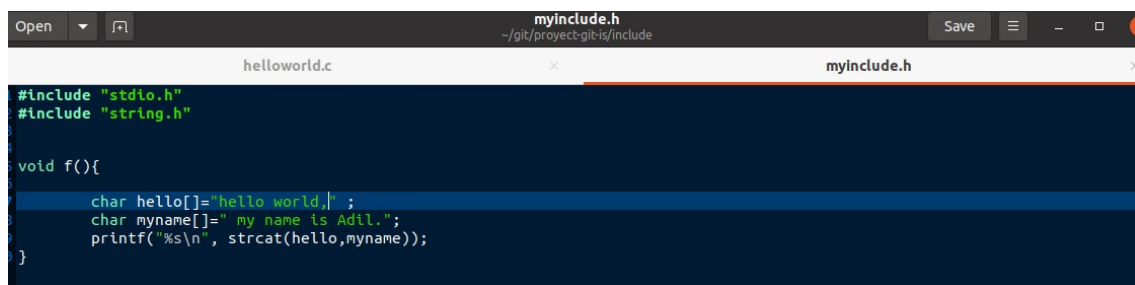
despues de haber compilado el codigo y ejecutado se genera el ejecutable dentro de nuestro repositorio con posibilidad de agregar al "stagin" aunque no tendria sentido guardar todos los ejecutables,ya que nuestro fin general del repositorio es crear un repositorio con nuestro codigo limpio y legible por todos los participantes .



8.1. crear nueva rama "branchA":

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git branch branchA
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

8.2. modificar nuestro código :



```
myinclude.h
~/git/proyect-git-is/include

#include "stdio.h"
#include "string.h"

void f(){
    char hello[]="hello world, ";
    char myname[]=" my name is Adil.";
    printf("%s\n", strcat(hello,myname));
}
```

9. confirmar los cambios :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git add .
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "now with strcat"
[master a0ab22b] now with strcat
2 files changed, 10 insertions(+)
create mode 100755 include/helloworld
create mode 100644 include/myinclude.h
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

10.1. apuntar a la rama master :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git checkout master
Already on 'master'
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```



10.2.modificar nuestro codigo helloworld.c :

```
1 /*****  
2  
3     author : Adil , Hirchi  
4     date : 22 sept 2020  
5     profesora : Aurora Ramirez quesada  
6     asignatura : ingeniería de software  
7 *****/  
8  
9  
10 #include "myinclude.h"  
11  
12  
13  
14 int main()  
15 {  
16  
17     f() ;  
18  
19 }
```

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git status  
On branch master  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
        modified:   helloworld.c  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

10.3.confirmar los cambios :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "now with author and date"  
[master df07fd0] now with author and date  
1 file changed, 9 insertions(+)  
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```



11.fusionar la rama master con branchA :

```

adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "now with author and date"
[master df07fd0] now with author and date
1 file changed, 9 insertions(+)
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git checkout branchA
Switched to branch 'branchA'
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git log --oneline --graph --all
* df07fd0 (master) now with author and date
* a0ab22b now with strcat
* 1bbd543 (HEAD -> branchA) initial commit
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git merge branchA master
Updating 1bbd543..df07fd0
Fast-forward
 include/helloworld | Bin 0 -> 16816 bytes
 include/helloworld.c | 9 ++++++++
 include/myinclude.h | 10 ++++++++
 3 files changed, 19 insertions(+)
 create mode 100755 include/helloworld
 create mode 100644 include/myinclude.h
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git log --oneline --graph --all
* df07fd0 (HEAD -> branchA, master) now with author and date
* a0ab22b now with strcat
* 1bbd543 initial commit
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$

```

12. fusionar dos ramas creando conflicto :

paso 1 : modificamos por ejemplo el apellido en el comentario y guardamos los cambios en el fichero hellowrold.c y lo confiramos con apuntador "branchA" y titulo "cambiar author"

commit -a -m "cambiar author"

paso 2 : modificamos el mismo fichero hellowrold.c y la misma linea de codigo que en nuestro caso ha sido comentario (apellido del author con segundo apellido por ejemplo) .

guardamos el ficheor y lo añadimos confirmando los cambios creando nueva rama con apuntado "conflicto" tal y como muestra la siguiente captura:



```
adil@adil-HP-Laptop-15s-fq1xxx: ~/git/proyect-git-is/include
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git log --oneline --graph --all
* e486082 (HEAD -> branchA, conflicto) now changing author name
* df07fd0 (master) now with author and date
* a0ab22b now with strcat
* 1bbd543 initial commit
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

despues de haber hecho el commit el sistema git nos altera haber un conflicto entre dos cambios de codigo para corregir tal y como muestra la siguiente captura y se refleja el conflicto tambien en nuestro fichero :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "changing title"
[conflicto3 38699b0] changing title
1 file changed, 3 insertions(+), 3 deletions(-)
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git checkout master
Switched to branch 'master'
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "add chang title"
[master 4e7ec89] add chang title
1 file changed, 1 insertion(+), 1 deletion(-)
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git merge conflicto3
Auto-merging include/helloworld.c
CONFLICT (content): Merge conflict in include/helloworld.c
Automatic merge failed; fix conflicts and then commit the result.
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```

paso 3 : resolver el conflicto



```
1 /*****  
2 <<<<<< HEAD  
3  
4     author : Adil , Hirchi el guerouj...changin name to commit  
5 =====  
6  
7     author : Adil  
8 >>>>>> conflicto3  
9     date : 22 sept 2020  
10    profesora : Aurora Ramirez  
11    asignatura : ingenieria de informatica rama software  
12 *****/  
13  
14  
15 #include "myinclude.h"  
16  
17  
18  
19 int main(){  
20  
21     f() ;  
22  
23  
24 }
```

para resolver el conflicto nos dirigimos al fichero donde nos asigna git el conflicto y corregimos el doble conflicto quedando solo con el código o comentario que deseamos tal y como muestra la figura (quedando solo con apellido de un único autor):

paso 3 : guardamos y confirmamos los cambios en la rama “resolved conflict” :

git commit -a -m “conflicto resuelto”

paso 4 : comprobar el estado de git

git status



**paso 5 : fusionar las dos ramas anteriores donde
ocurrio el conflicto .**

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git commit -a -m "conflicto resuelto"
[master 22ca892] conflicto resuelto
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git merge conflicto3
Already up to date.
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git log --oneline --graph --all
* 22ca892 (HEAD -> master) conflicto resuelto
```

paso 6 : comprobar la fusion de las dos ramas

**en la siguiente figura observamos todos los pasos
citados anteriormente y notamos como la rama se une
fusionando las dos ramas en una unica rama
resolviendo el conflicto**

```
adil@adil-HP-Laptop-15s-fq1xxx:~/gitslycs/include$ git commit -a -m "resolved co
nflict"
[conflicto 45223f0] resolved conflict
adil@adil-HP-Laptop-15s-fq1xxx:~/gitslycs/include$ git status
On branch conflicto
nothing to commit, working tree clean
adil@adil-HP-Laptop-15s-fq1xxx:~/gitslycs/include$ git merge branchA conflicto
Already up to date.
adil@adil-HP-Laptop-15s-fq1xxx:~/gitslycs/include$ git log --oneline --decorate
--graph --all
* 45223f0 (HEAD -> conflicto) resolved conflict
| \
| * 712406a (branchA) cambiar author
* | f0860e3 conflicto2
|/
```

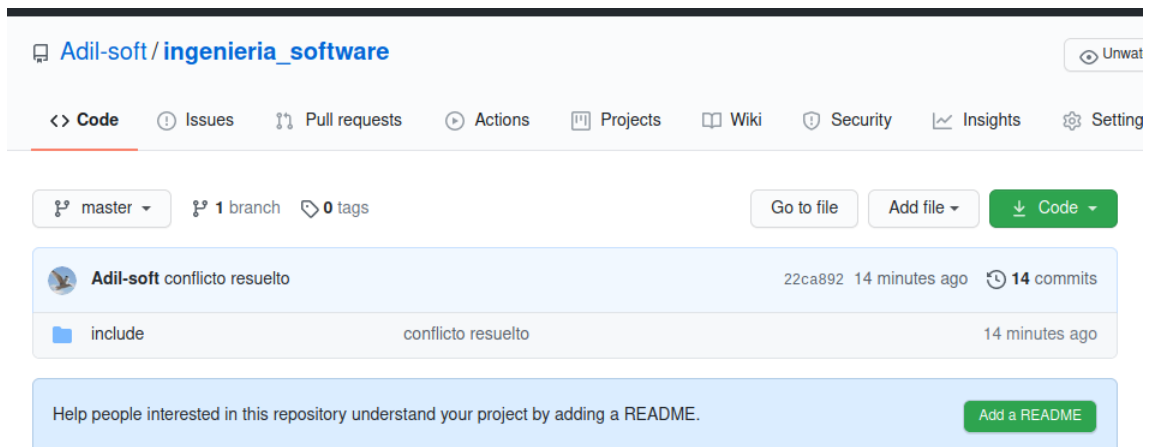
13.subir proyecto a github :

**13.1.subir el repositorio desde nuestro terminal a
github :**



```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git remote add origin https://github.com/Adil-soft/ingenieria_software.git
fatal: remote origin already exists.
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git push -u origin master
Username for 'https://github.com': adil-soft
Password for 'https://adil-soft@github.com':
Enumerating objects: 58, done.
Counting objects: 100% (58/58), done.
Delta compression using up to 8 threads
Compressing objects: 100% (44/44), done.
Writing objects: 100% (58/58), 6.97 KiB | 2.32 MiB/s, done.
Total 58 (delta 23), reused 0 (delta 0)
remote: Resolving deltas: 100% (23/23), done.
To https://github.com/Adil-soft/ingenieria_software
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

13.2.comprobar nuestro proyecto en nuestra cuenta github :



observamos nuestro proyecto incluye ya dentro de nuestro repositorio .



Adil-soft / ingenieria_software

Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master ingenieria_software / include / Go to file Add file

Adil-soft conflicto resuelto 22ca892 15 minutes ago History

..		
helloworld	now with strcat	1 hour ago
helloworld.c	conflicto resuelto	15 minutes ago
helloworld.h	initial commit	2 hours ago
myinclude.h	now with strcat	1 hour ago

14. bajar un proyecto desde Github :

```
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$ git clone https://github.com/Adil-soft/ingenieria_software.git
Cloning into 'ingenieria_software'...
remote: Enumerating objects: 58, done.
remote: Counting objects: 100% (58/58), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 58 (delta 23), reused 58 (delta 23), pack-reused 0
Unpacking objects: 100% (58/58), 6.95 KiB | 647.00 KiB/s, done.
adil@adil-HP-Laptop-15s-fq1xxx:~/git/proyect-git-is/include$
```



despues de haber invocado “clone” podemos acceder a nuestro proyecto y ubicarlo en nuestra computadora .de forma similar al repositorio que disponemos en github . Tal y como muestra la siguiente captura :

