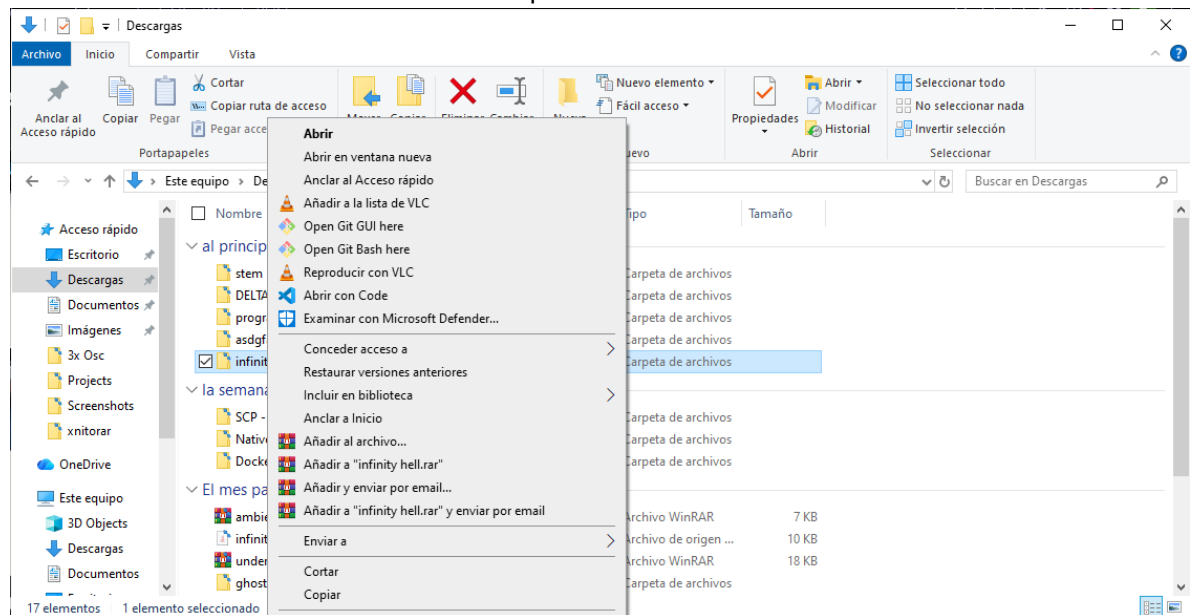


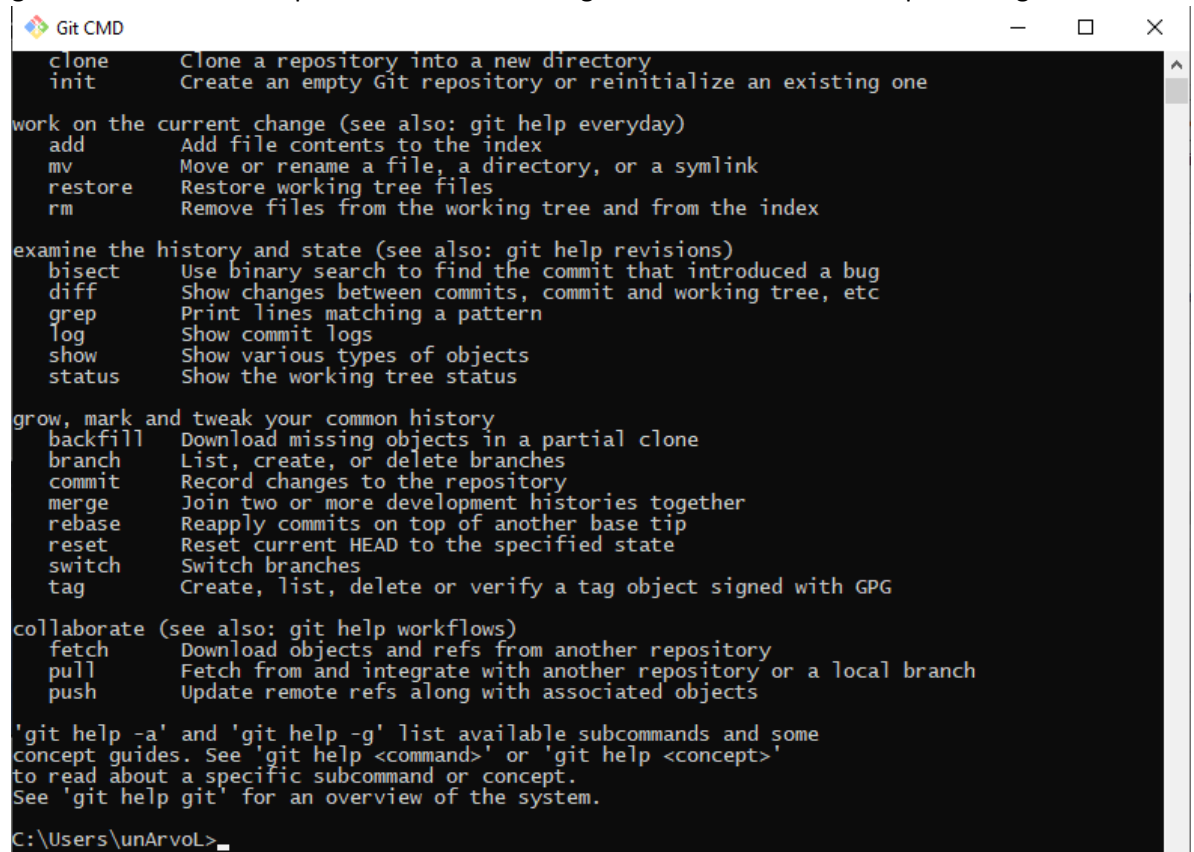
1. Instalacion de Git:

Aquí, yo aplico la del yo no lo instalo porque ya lo tengo, porque si lo hago me arrojaría error a lo ultimo entonces por eso no lo instalo de nuevo



2. Comandos básicos de la terminal:

git: este comando sirve para obtener un listado genérico de los comandos que tiene git



```
Git CMD
clone      Clone a repository into a new directory
init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
add        Add file contents to the index
mv         Move or rename a file, a directory, or a symlink
restore    Restore working tree files
rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
bisect     Use binary search to find the commit that introduced a bug
diff       Show changes between commits, commit and working tree, etc
grep       Print lines matching a pattern
log        Show commit logs
show       Show various types of objects
status     Show the working tree status

grow, mark and tweak your common history
backfill   Download missing objects in a partial clone
branch     List, create, or delete branches
commit     Record changes to the repository
merge      Join two or more development histories together
rebase     Reapply commits on top of another base tip
reset      Reset current HEAD to the specified state
switch     Switch branches
tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
fetch      Download objects and refs from another repository
pull       Fetch from and integrate with another repository or a local branch
push       Update remote refs along with associated objects

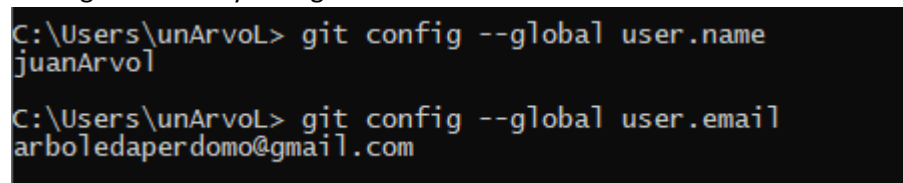
'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

C:\Users\unArvoL>
```

3. Configuración de git:

git config --global user.name/ user.email:

estos comandos se usan para definir de forma global en el equipo, que como mencione ya lo tengo instalado y configurado.

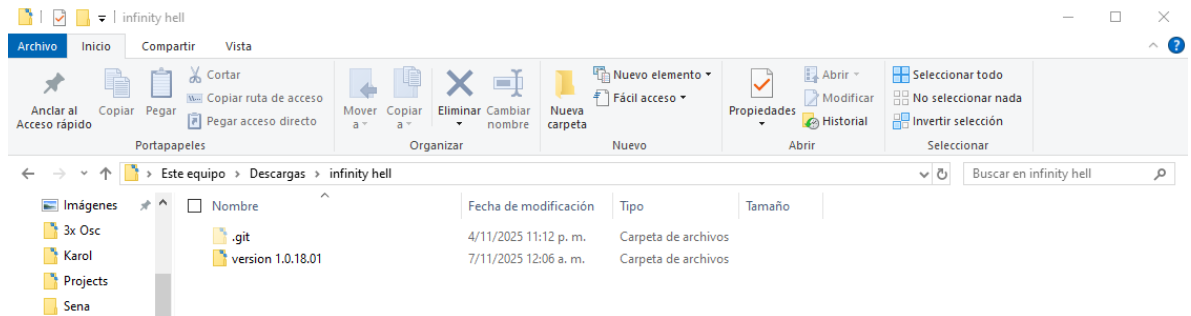


```
C:\Users\unArvoL> git config --global user.name
juanArvoL

C:\Users\unArvoL> git config --global user.email
arboledaperdomo@gmail.com
```

4. git init:

Este comando se usa para inicializar el proceso de carga de un repositorio git



5. Ramas en git:

Son diferenciaciones en el flujo del repositorio y puede llamarse cualquier forma

```
C:\Users\unArvoL>git init
Initialized empty Git repository in C:/Users/unArvoL/.git/

C:\Users\unArvoL>git branch -m arvo1

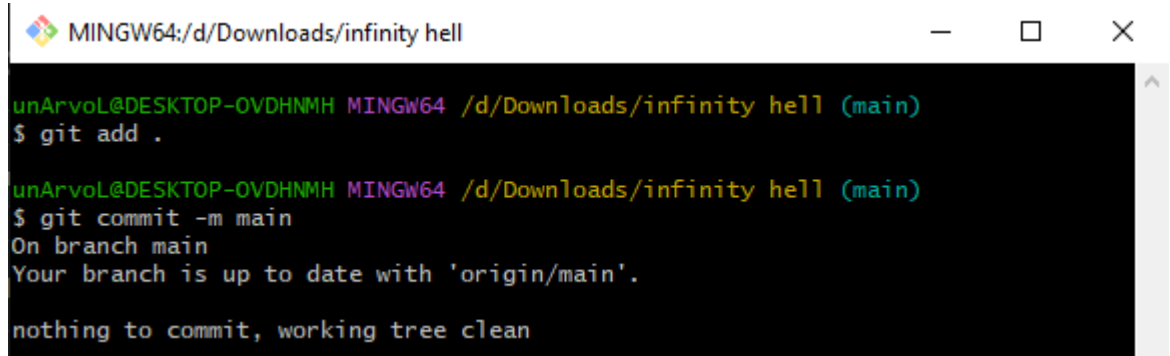
C:\Users\unArvoL>git branch -m main
```

6. Git add y git commit

Estos comandos son el principal medio de control del repositorio git.

El git add sirviendo para indicar que archivo quiero cargar.

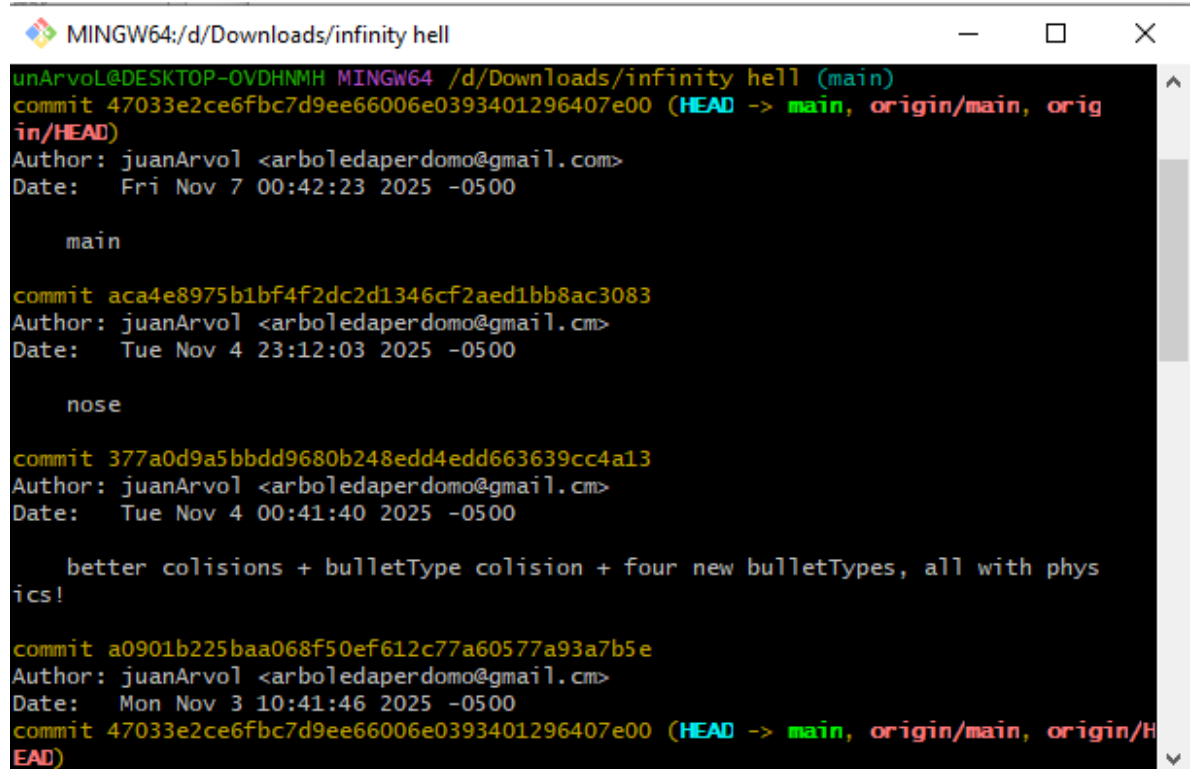
El git commit sirviendo para indicar el estado del archivo a cargar.



7. Git log y git status

El git log muestra todos los registros y cambios que se han hecho en el repositorio desde la rama que se trabaja.

El git status muestra el estado actual de los ficheros que tiene el proyecto para así comprobar su integridad



```
MINGW64:/d/Downloads/infinity hell
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
commit 47033e2ce6fbc7d9ee66006e0393401296407e00 (HEAD -> main, origin/main, origin/H
in/HEAD)
Author: juanArvo1 <arboledaperdomo@gmail.com>
Date:   Fri Nov 7 00:42:23 2025 -0500

    main

commit aca4e8975b1bf4f2dc2d1346cf2aed1bb8ac3083
Author: juanArvo1 <arboledaperdomo@gmail.cm>
Date:   Tue Nov 4 23:12:03 2025 -0500

    nose

commit 377a0d9a5bbdd9680b248edd4edd663639cc4a13
Author: juanArvo1 <arboledaperdomo@gmail.cm>
Date:   Tue Nov 4 00:41:40 2025 -0500

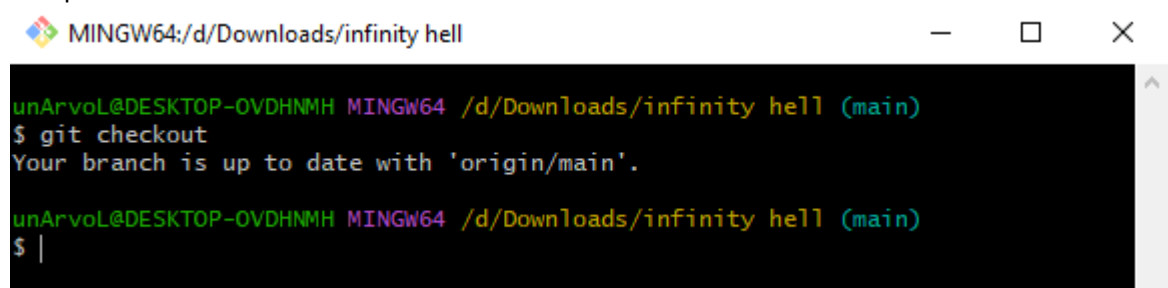
    better colisions + bulletType colision + four new bulletTypes, all with phys
ics!

commit a0901b225baa068f50ef612c77a60577a93a7b5e
Author: juanArvo1 <arboledaperdomo@gmail.cm>
Date:   Mon Nov 3 10:41:46 2025 -0500
commit 47033e2ce6fbc7d9ee66006e0393401296407e00 (HEAD -> main, origin/main, origin/H
EAD)
```

8. Git checkout y git reset

El git checkout funciona como un ctrl + z ya que permite el acceso a lo que viene siendo la versión anterior a la que no este guardada

El git reset directamente vuelve a la versión anterior de un archivo que ya este subido en el repositorio



```
MINGW64:/d/Downloads/infinity hell
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git checkout
Your branch is up to date with 'origin/main'.

unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ |
```

9. Git alias

el git alias funciona para crear “funciones” dentro del git, permitiendo así el llamado de un alias y así mismo ejecutar el comando asociado a el

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git config --global alias.a "log"

unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git a
commit 47033e2ce6fbc7d9ee66006e0393401296407e00 (HEAD -> main, origin/main, origin/HEAD)
Author: juanArvoL <arboledaperdomo@gmail.com>
Date:   Fri Nov 7 00:42:23 2025 -0500

    main

commit aca4e8975b1bf4f2dc2d1346cf2aed1bb8ac3083
Author: juanArvoL <arboledaperdomo@gmail.com>
Date:   Tue Nov 4 23:12:03 2025 -0500

    nose

commit 377a0d9a5bbdd9680b248edd4edd663639cc4a13
Author: juanArvoL <arboledaperdomo@gmail.com>
Date:   Tue Nov 4 00:41:40 2025 -0500
```

10. Git ignore

El git ignore es un tipo de archivo usado para ignorar ficheros que no nos interesen al momento de hacer depurado de los archivos que esten o no cargados en el repositorio

11. Git diff

El git diff es un comando utilizado para poder visualizar los cambios que hay entre la versión actual cargada en el repositorio comparada con la que nosotros tenemos en el pc

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git diff

unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$
```

12. Desplazamiento en una rama

Esto es una de las tantas funcionalidades que tiene git, aca lo que se consigue es el acceso a versiones pasadas cargadas en el fichero local del pc siempre y cuando esten cargadas y subidas.

13. Git reset --hard y git reflog

El git reset --hard es un comando para forzar el inicio de nuestro fichero y así mismo eliminar todo lo que fue hecho después del fichero al que apuntamos el reset --hard.

El git reflog es un comando que muestra absolutamente todas las movidas que se han hecho dentro del fichero en el que estamos trabajando.

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git reflog
47033e2 (HEAD -> main, origin/main, origin/HEAD) HEAD@{0}: commit: main
aca4e89 HEAD@{1}: commit: nose
377a0d9 HEAD@{2}: commit: better colisions + bulletType colision + four new bulletTypes, all with physics!
a0901b2 HEAD@{3}: commit: sistema de colisiones de bala unitaria implementado pd , ya le entendi + al juego xd
ee4a3b1 HEAD@{4}: commit: valiendo madres pero ya le entendi al flujo del juego xd
32adbe5 HEAD@{5}: commit: todo vale madres pero ya no tanto
4e6fa37 HEAD@{6}: commit: todo vale madres pero ahora entiendo mas xd
aa2100e HEAD@{7}: commit: todo sigue valiendo madres pero al menos ya comprendo mas el juego xd
5df1104 HEAD@{8}: commit: todo vale madres pero aca subo pa david
8f935f0 HEAD@{9}: commit: life implement
c93e954 HEAD@{10}: commit: bullets update
b95c2a6 HEAD@{11}: commit: nose
d571a22 HEAD@{12}: commit: aim and physics correct
b17a074 HEAD@{13}: commit: a
275367a HEAD@{14}: commit: better colisions xd
37647c9 HEAD@{15}: commit: mejores colisiones y masa
608bc18 HEAD@{16}: commit: version 1.0.17.34
```

14. Git tag

El git tag es un comando clasificativo que nosotros definimos para sobre nombrar nuestros ficheros hechos y así mismo tener una identificación que nosotros definamos.

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git tag ultimaVersion

unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$
```

15. Git branch y git switch

El git branch es un comando usado para la creación de ramas.

El git switch es para intercambiar la rama en la que se está trabajando.

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git branch dav

unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (main)
$ git switch dav
Switched to branch 'dav'
```

16. Git merge

El git merge es un comando para mezclar 2 versiones en un fichero y así mismo compilar ambas versiones en una sola rama.

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (dav)
$ git merge main
Already up to date.
```

17. resoluciones de conflictos en git.

La resolucion de conflictos en git se da cuando varias ramas modifican un mismo fichero y git reconoce esto como un conflicto, siendo asi que para que funcione el comando que se este aplicando hay que definir que version de rama se quiere definir.

18. git stash.

El comando git stash permite guardar temporalmente los cambios hechos en un fichero, ya con git stash pop se hace llamado a este guardado. Git stash drop elimina la version guardada.

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (dav)
$ git stash
No local changes to save
```

19. Reintegracion de ramas de ramas en git.

Esto permite la reestructuracion de ramas, permitiendo el salvado y reintegracion de lo que se quiera rescatar de las distintas ramas que se tienen y añadirlos a la rama principal.

20. Eliminacion de ramas.

La eliminacion de ramas permite que los cambios temporales almacenados dentro de una rama se puedan eliminar, para asi mismo dejar unicamente las ramas que se deseen.

Esto se logra con el git branch -p

21. Introduccion a github.

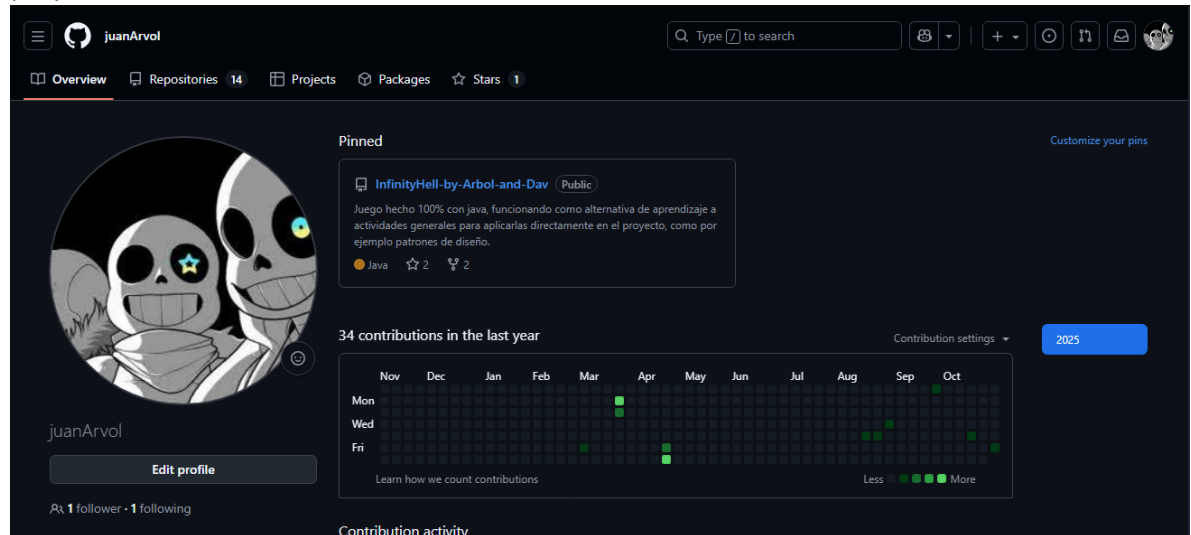
Aca se hace la aclaracion del que git y github no son lo mismo, siendo que git es un sistema de almacenado de ficheros por medio de ramas mientras que github es una plataforma para exportar dichos ficheros y tenerlos almacenados en la nube.

22. Primeros pasos en github

Aca se nos hace crear una cuenta de github para asi mismo tener acceso a los servicios de github en la nube.

23. Repositorio personal:

Aca se nos hace crear nuestro primer repositorio personal para hacer carga de nuestro proyecto en cuestion



24. Local y remoto:

Aca se hace la explicacion de los estados de carga del "proyecto"

Local: cuando nuestro proyecto aun sigue siendo accesible unicamente desde nuestro dispositivo pc.

Remoto: cuando nuestro proyecto es accesible desde un repositorio de github.

25. Autenticacion SSH en github:

Aca se explica que la autenticacion SSH es la forma en la que github comprueba que quien esta manejando el git tiene acceso al repositorio remoto.

Para generar una se hace uso del comando: `ssh-keygen -t ed25519 -C "gmail de quien va a acceder al repositorio en github"`

```
C:\Users\unArvoL>ssh-keygen -t ed25519 -C "arboledaperdomo@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\unArvoL/.ssh/id_ed25519): id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa
Your public key has been saved in id_rsa.pub
The key fingerprint is:
SHA256:HRkDh4M4h25+QXNH1Bc0ozA9SMQBfN34K3jDwt6Jp1E arboledaperdomo@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|+. *BX*+o ..|
|+ * B+*=O+ .|
|. + + O+O ..|
|O . . . .|
|o .SoE .|
|. . . * .|
|. . = = .|
|. + O|
|.o|
+-----[SHA256]-----+
```

26. Repositorio proyecto:

Aca se nos explica como preparar el proyecto para ser vinculado y ser subido en el repositorio remoto del github.

27. Git remote:

El git remote es el comando por el cual se define a que repositorio del github se hara carga de nuestro proyecto local, se hace mediante el comando: git remote add origin "direccion del repositorio github"

```
C:\Users\unArvoL>git remote add origin https://github.com/juanArvoL/InfinityHell-by-Arbol-and-Dav
C:\Users\unArvoL>_
```

28. Subida de un proyecto a github:

Aca se explica que para hacer carga y subida del proyecto a github se es necesario que el local este commiteado la rama que se quiera subir.

29. Git fetch y git pull:

El git pull es el comando con el que se sube el archivo seleccionado del proyecto local y se hace mediante el git pull -u origin "nRama."

El git fetch es un comando que funciona para ver el estado de los archivos que se encuentran en el proyecto.

```
C:\Users\unArvoL>git fetch
remote: Enumerating objects: 432, done.
remote: Counting objects: 100% (432/432), done.
remote: Compressing objects: 100% (257/257), done.
remote: Total 432 (delta 203), reused 342 (delta 113), pack-reused 0 (from 0)
Receiving objects: 100% (432/432), 435.19 KiB | 794.00 KiB/s, done.
Resolving deltas: 100% (203/203), done.
From https://github.com/juanArvoL/InfinityHell-by-Arbol-and-Dav
* [new branch]      main      -> origin/main
```

30. git clone:

El git clone es un comando para clonar un repositorio por medio de comandos y de la terminal.

```
C:\Users\unArvoL>.git clone https://github.com/juanArvoL/InfinityHell-by-Arbol-and-Dav.git
Cloning into 'InfinityHell-by-Arbol-and-Dav'...
remote: Enumerating objects: 432, done.
remote: Counting objects: 100% (432/432), done.
remote: Compressing objects: 100% (257/257), done.
remote: Total 432 (delta 203), reused 342 (delta 113), pack-reused 0 (from 0)
Receiving objects: 100% (432/432), 435.19 KiB | 852.00 KiB/s, done.
Resolving deltas: 100% (203/203), done.
C:\Users\unArvoL>
```

31. git push:

El git push es el comando usado para subir/cargar el proyecto ya commiteado al repositorio github y esto se logra con git push -u origin "rama"

```
unArvoL@DESKTOP-OVDHNMH MINGW64 /d/Downloads/infinity hell (dav)
$ git push -u origin dav
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: This repository moved. Please use the new location:
remote:   https://github.com/juanArvoL/InfinityHell-by-Arbol-and-Dav.git
remote:
remote: Create a pull request for 'dav' on GitHub by visiting:
remote:   https://github.com/juanArvoL/InfinityHell-by-Arbol-and-Dav/pull/new/dav
remote:
To https://github.com/juanArvoL/asociados-main.git
* [new branch]      dav -> dav
branch 'dav' set up to track 'origin/dav'.
```

32. "fork" en github

El "fork" es una clonación de un repositorio de un github a nuestro propio github para así mismo tener, nuestro propio github para trabajar en el proyecto.

33. Flujo colaborativo en github:

Esto describe el comportamiento que se a de seguir cuando ocurre el "fork" siendo que lo ideal es hacer un merge del repositorio al que se iba a hacer carga inicialmente y actualizarlo con lo que sea que hayamos modificado, solicitando los permisos para así mismo poder acceder a ese repositorio.

34. Pull request(PR) en github:

Esto es la aceptación del flujo colaborativo en github, siendo que acá es donde se despacha la solicitud de cambio/merge del repositorio en el que no tenemos permisos para así solicitar que nuestro aporte sea añadido al repositorio principal.

35. ejercicio practico:

Acá se solicita que accedamos al repositorio de él para así mismo hacer un pull request y hacer parte del repositorio de él.

36. resolución de conflictos de pull request:

Esto es una clase explicando los distintos problemas que pueden ocurrir al hacer un pull request y así mismo explicando las diferentes formas de solucionar el conflicto que tenga el pull request.

37. sincronización de un fork en github:

Esta es una opción que trae el github para sincronizar los repositorios que clonemos en el propio github para así mismo tener nuestro fork actualizado/sincronizado al del repositorio original.

38. Markdown en github:

Acá se explica el uso de los readme.md para documentar los repositorios que tengamos y así llevar un registro adicional de lo que sea que modifiquemos dentro de él.

39. Herramientas gráficas (GUI) para git y github:

Acá se explica la existencia de herramientas que facilitan el uso de git y github, aconsejando que el uso de estas es a nivel personal y que cada quien decide que usar, siempre y cuando ya comprenda lo que significa tanto git como github.

40. Git y github "flow":

Acá se explica el flujo que se maneja el orden de desarrollo ya sea dentro de un proyecto en grupo o en uno personal, independientemente a esto se explica que lo ideal es llevar un orden y seguir unos estándares para usar git y github y así mismo, se habla y explica sobre el git flow y el github flow.

41. Ejemplo gitflow:

Aca muestra un ejemplo del como se maneja el git flow y el github flow de forma sencilla y intuitiva para asi mostrar como se maneja el flujo apartir de esta funcionalidad. Explicando el nuevo flujo y orden que se a de llevar al manejar el git flow.

42. git cherry-pick y git rebase:

El git cherry-pick es un comando que funciona para traer un commit en especifico y trae justo los cambios hechos en dicho commit

El git rebase es un comando que funciona para “reiniciar” una rama a un punto especifico de un commit modificando la jerarquia que tienen las ramas y modificando asi el orden del repositorio.

43. Github pages y actions:

Aca se explica que github permite tambien si se sabe configurar el propio repositorio para asi mismo desplegar el proyecto que se tenga siempre y cuando este cumpla los requisitos para hacer uso de ese servicio de hosting totalmente gratis. Asi mismo se habla sobre github actions, y se explica que existe un api llamado de esta forma que permite el automatizar los repositorios y asi mismo poder definir que acciones queremos que se ejecuten por defecto y de forma autonoma en nuestro repositorio github.