

GIT + Hola mundo en OpenGL

Auxiliar N°2

CC3501 – Modelación y Computación Gráfica para Ingenieros

Diego Donoso

Contenidos de hoy

- GIT: Breve tutorial
- OpenGL - Shaders
- Ejemplo: Pirámides

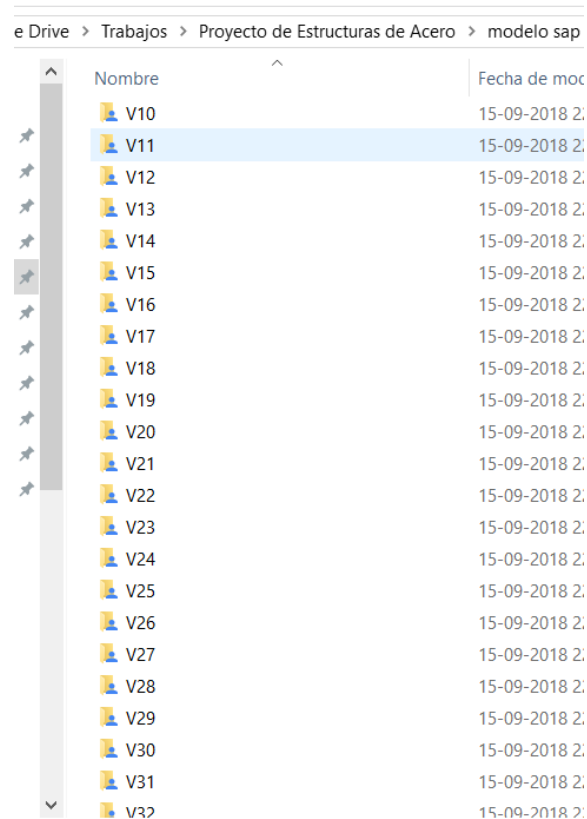
Git – Mini tutorial

- ¿Qué es el **control de versiones**, y por qué debería importarte? El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.
- Un método de control de versiones usado por mucha gente es copiar los archivos a otro directorio.

Fuente: <https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>

Git - Introducción

- Problemas de copiar archivos a un directorio distinto:



The screenshot shows a file explorer window with the path "e Drive > Trabajos > Proyecto de Estructuras de Acero > modelo sap". It displays a list of files named V10 through V32, each with a date of "15-09-2018 2:". The file V11 is highlighted. This illustrates a problem where files are copied to a new directory, but the original files remain, leading to redundancy and potential confusion.

Nombre	Fecha de mod
V10	15-09-2018 2:
V11	15-09-2018 2:
V12	15-09-2018 2:
V13	15-09-2018 2:
V14	15-09-2018 2:
V15	15-09-2018 2:
V16	15-09-2018 2:
V17	15-09-2018 2:
V18	15-09-2018 2:
V19	15-09-2018 2:
V20	15-09-2018 2:
V21	15-09-2018 2:
V22	15-09-2018 2:
V23	15-09-2018 2:
V24	15-09-2018 2:
V25	15-09-2018 2:
V26	15-09-2018 2:
V27	15-09-2018 2:
V28	15-09-2018 2:
V29	15-09-2018 2:
V30	15-09-2018 2:
V31	15-09-2018 2:
V32	15-09-2018 2:

- ¿Qué cambios hice en cada versión?
- ¿Qué archivos se cambiaron?
- (Pensando en un proyecto con varios colaboradores) ¿Quién hizo los cambios?
- ¿Cuándo se hicieron los cambios?
- ¿Qué pasa si el proyecto pesa mucho? ¿Conviene tener una carpeta completa por cada versión?

Git – La solución

- Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos.
- Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea.
- Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

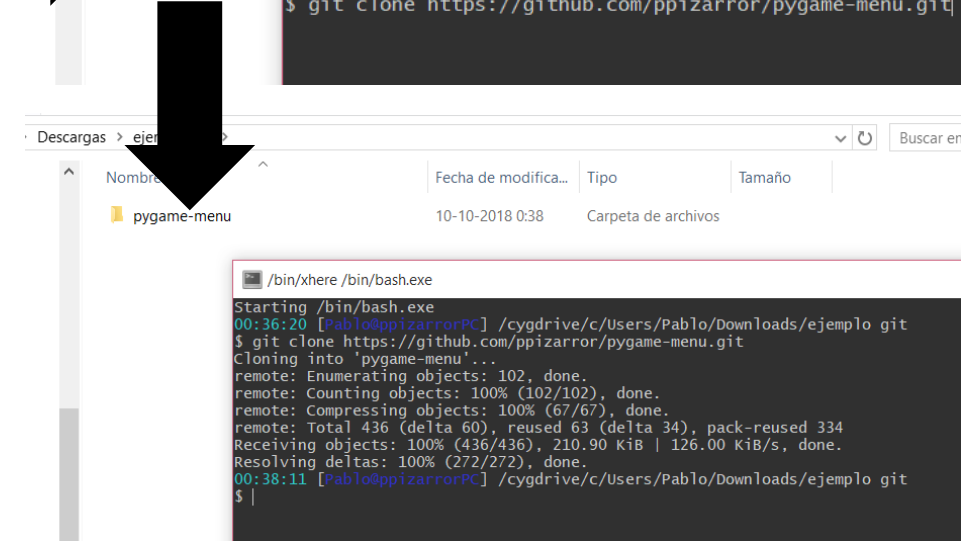
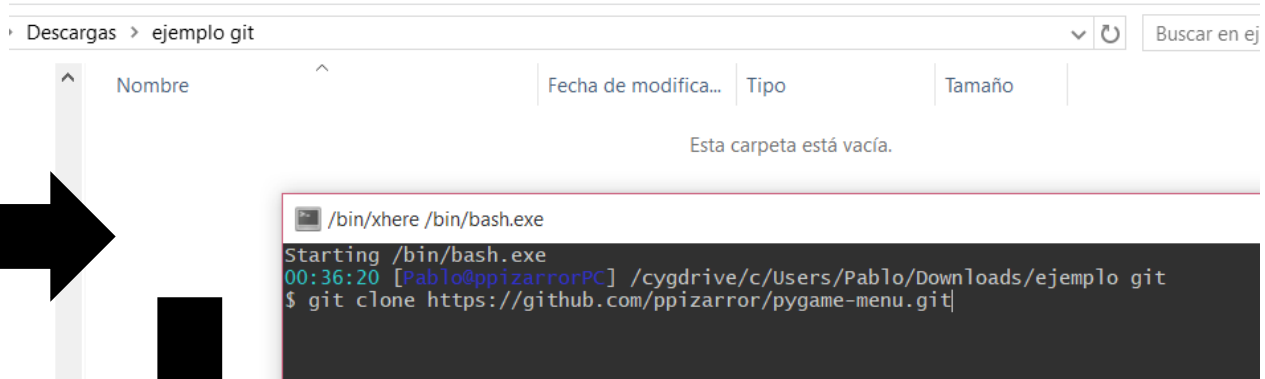
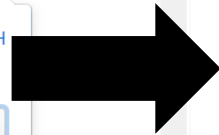
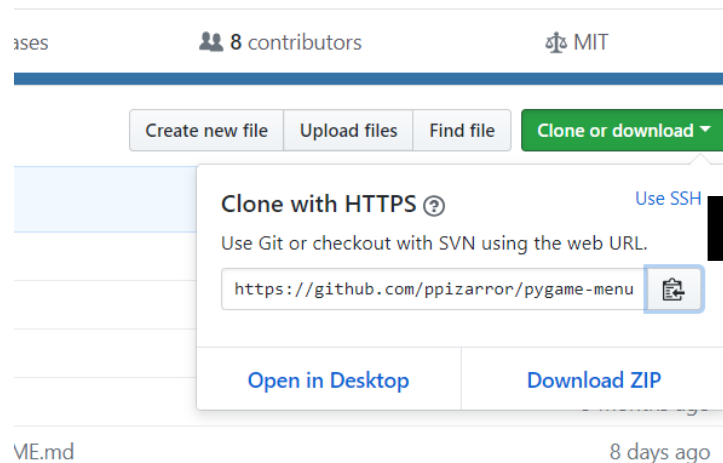
Fuente: <https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git>

Git – Creación de un repositorio

- Crea un directorio nuevo, ábrelo y ejecuta `git init` para crear un nuevo repositorio de git.
- También se puede crear un nuevo repositorio *remoto* en la nube, algunos servicios:
 - Github github.com/
 - Bitbucket <https://bitbucket.org>
 - Gitlab gitlab.com

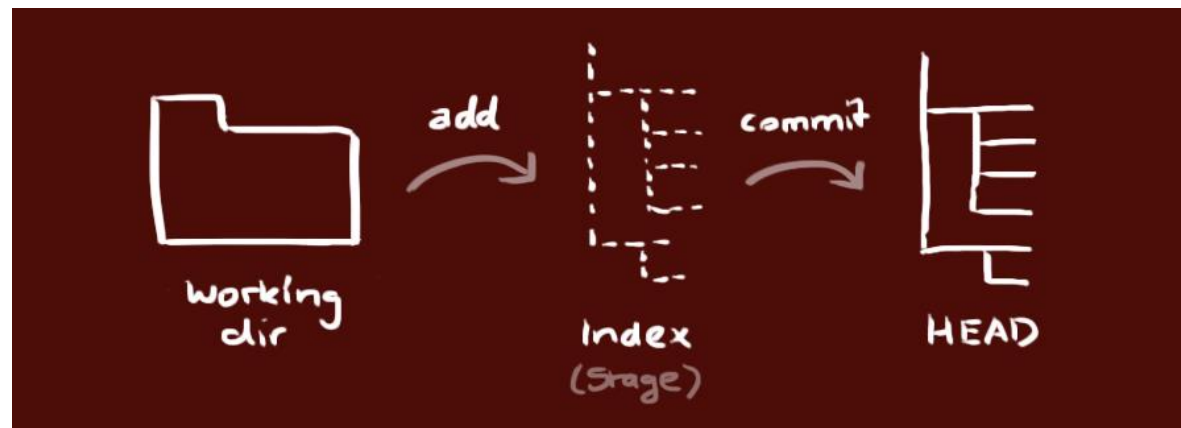
Git – Clonar un repositorio

- Crea una copia local del repositorio ejecutando `git clone /path/to/repository` desde una terminal.



Git – Flujo de trabajo

- Tu repositorio local esta compuesto por tres "árboles" administrados por git. El primero es tu **Directorio de trabajo** que contiene los archivos, el segundo es el **Index** que actua como una zona intermedia, y el último es el **HEAD** que apunta al último commit realizado.

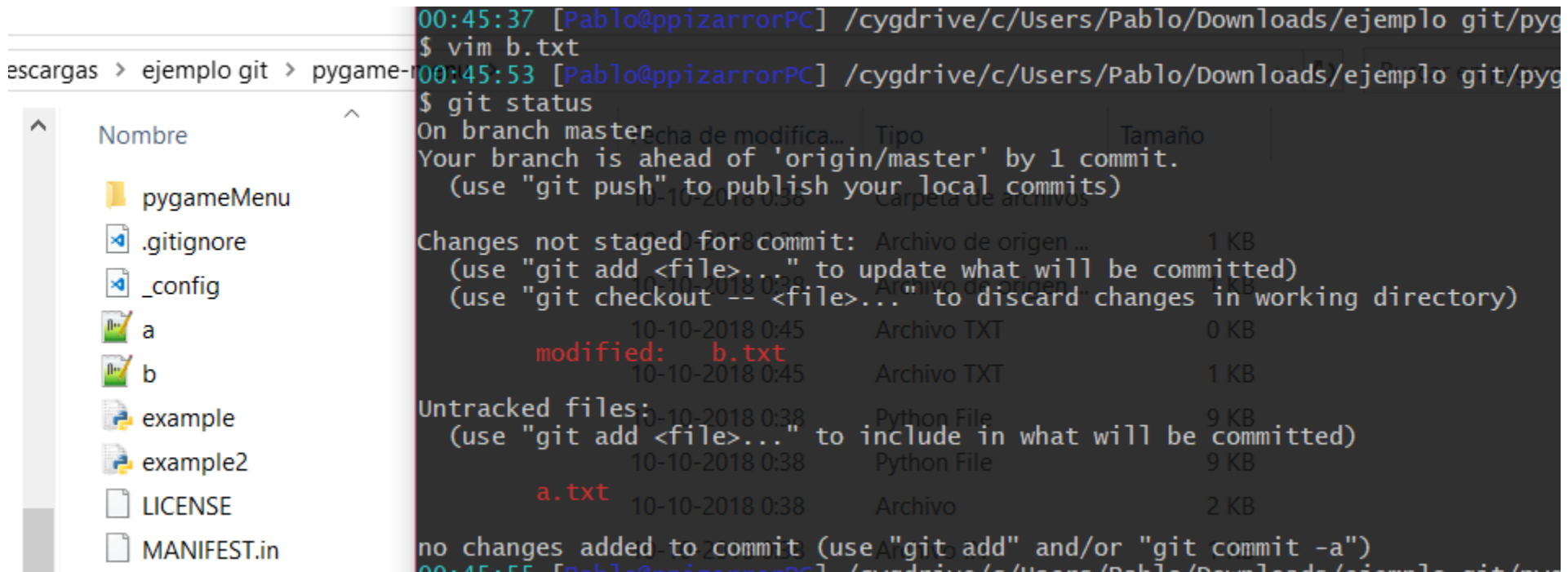


Git – Commits

- Un commit se puede entender como un paquete de cambios, en donde se pueden añadir archivos nuevos, agregar cambios en ciertos archivos o borrar archivos.
- Un commit tiene asociado un usuario, una fecha y un comentario.
- Para revisar qué cosas se han cambiado desde el último commit hasta la fecha se debe usar el comando `git status`

Git – Commits & status

- Ejemplo: Suponer que añadí un archivo `a.txt` y añadí tres líneas al archivo `b.txt`, `git status` dirá lo siguiente:



```
00:45:37 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Downloads/ejemplo git/pyg
$ vim b.txt
00:45:53 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Downloads/ejemplo git/pyg
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   b.txt

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

        a
        example
        example2
        LICENSE
        MANIFEST.in
        a.txt

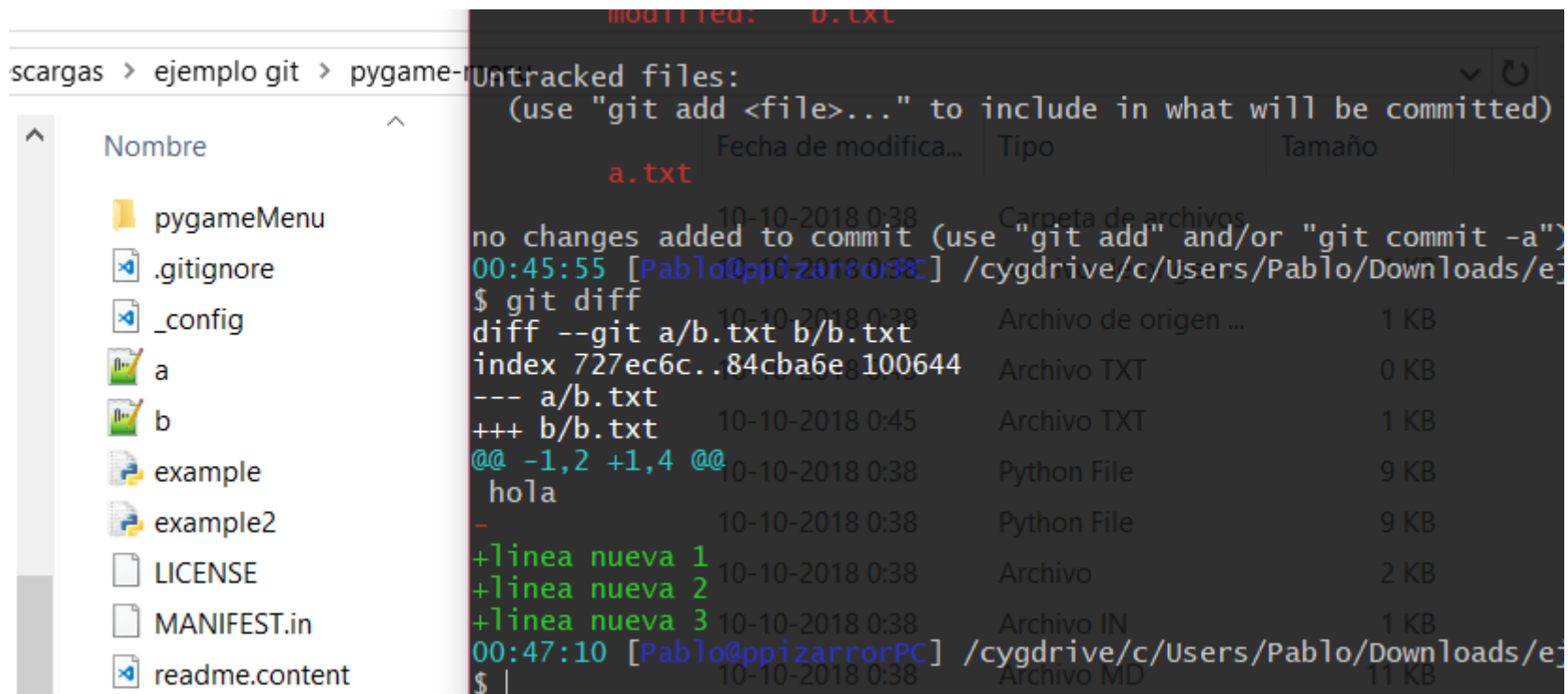
no changes added to commit (use "git add" and/or "git commit -a")
00:45:55 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Downloads/ejemplo git/pyg
```

File Explorer View:

- Nombre
- pygameMenu
- .gitignore
- _config
- a
- b
- example
- example2
- LICENSE
- MANIFEST.in

Git – Commits & status

- Ejemplo: Si quiero un detalle de los cambios nuevos se puede usar el comando `git diff`.



The image shows a file explorer window on the left and a terminal window on the right. The file explorer displays the contents of a directory named 'ejemplo git', which includes a folder 'pygameMenu' and several files: '.gitignore', '_config', 'a', 'b', 'example', 'example2', 'LICENSE', 'MANIFEST.in', and 'readme.content'. The terminal window shows the output of the 'git diff' command, which compares the current state of the repository with the last commit. The output indicates that there are no changes added to the commit, but it also shows the details of the files in the repository, including their names, sizes, and types.

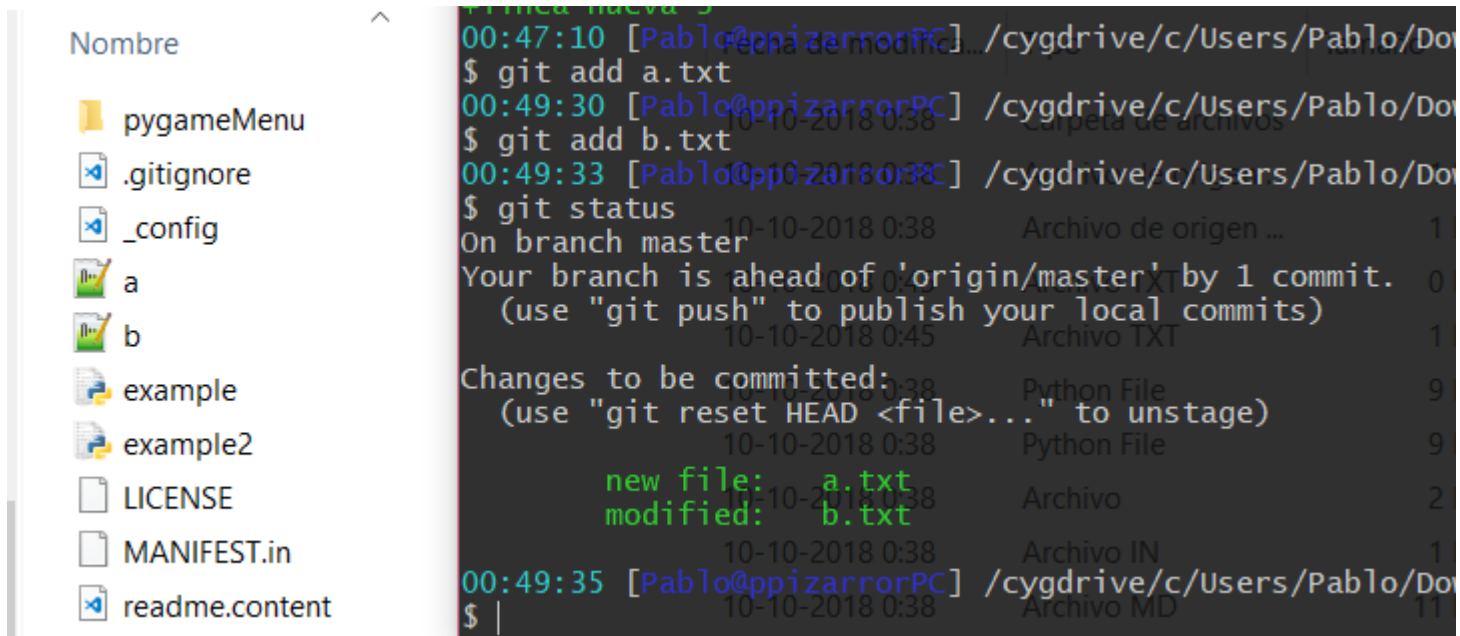
```
Untracked files:
(use "git add <file>..." to include in what will be committed)
a.txt

no changes added to commit (use "git add" and/or "git commit -a")
00:45:55 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Downloads/ejemplo git
$ git diff
diff --git a/b.txt b/b.txt
index 727ec6c..84cba6e 100644
--- a/b.txt
+++ b/b.txt
@@ -1,2 +1,4 @@
 hola
-
+linea nueva 1
+linea nueva 2
+linea nueva 3
00:47:10 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Downloads/ejemplo git
$
```

Nombre	Fecha de modifica...	Tipo	Tamaño
a.txt	10-10-2018 0:38	Archivo de origen ...	1 KB
b	10-10-2018 0:38	Archivo TXT	0 KB
b	10-10-2018 0:45	Archivo TXT	1 KB
example	10-10-2018 0:38	Python File	9 KB
example2	10-10-2018 0:38	Python File	9 KB
linea nueva 1	10-10-2018 0:38	Archivo	2 KB
linea nueva 2	10-10-2018 0:38	Archivo IN	1 KB
linea nueva 3	10-10-2018 0:38	Archivo MD	11 KB

Git – Commits & status

- Para añadir cambios al commit usar el comando `git add <filename>`, se pueden usar wildcards (*) o bien añadir todos los archivos nuevos/modificados con `git add --all`



The image shows a file explorer on the left and a terminal window on the right. The file explorer lists files: pygameMenu, .gitignore, _config, a, b, example, example2, LICENSE, MANIFEST.in, and readme.content. The terminal window shows the following commands and output:

```
00:47:10 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Do
$ git add a.txt
00:49:30 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Do
$ git add b.txt
00:49:33 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Do
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   a.txt
        modified:   b.txt
00:49:35 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Do
$ |
```

Lo que dice esta terminal es que el nuevo commit contiene un nuevo archivo a.txt y uno modificado b.txt

Git – Commits & status

- Para cerrar el commit se debe utilizar el comando `git commit -m "Commit message"` con esto se añade un comentario al paquete de cambios.

The image shows a terminal window with the following commands and output:

```

00:49:30 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Down
$ git add b.txt
00:49:33 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Down
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   a.txt
        modified:  b.txt

00:49:35 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Down
$ git commit -m "Mi nuevo commit"
[master 8a70e98] Mi nuevo commit
 2 files changed, 3 insertions(+), 1 deletion(-)
 create mode 100644 a.txt

00:51:54 [Pablo@ppizarrorPC] /cygdrive/c/Users/Pablo/Down
$ git push

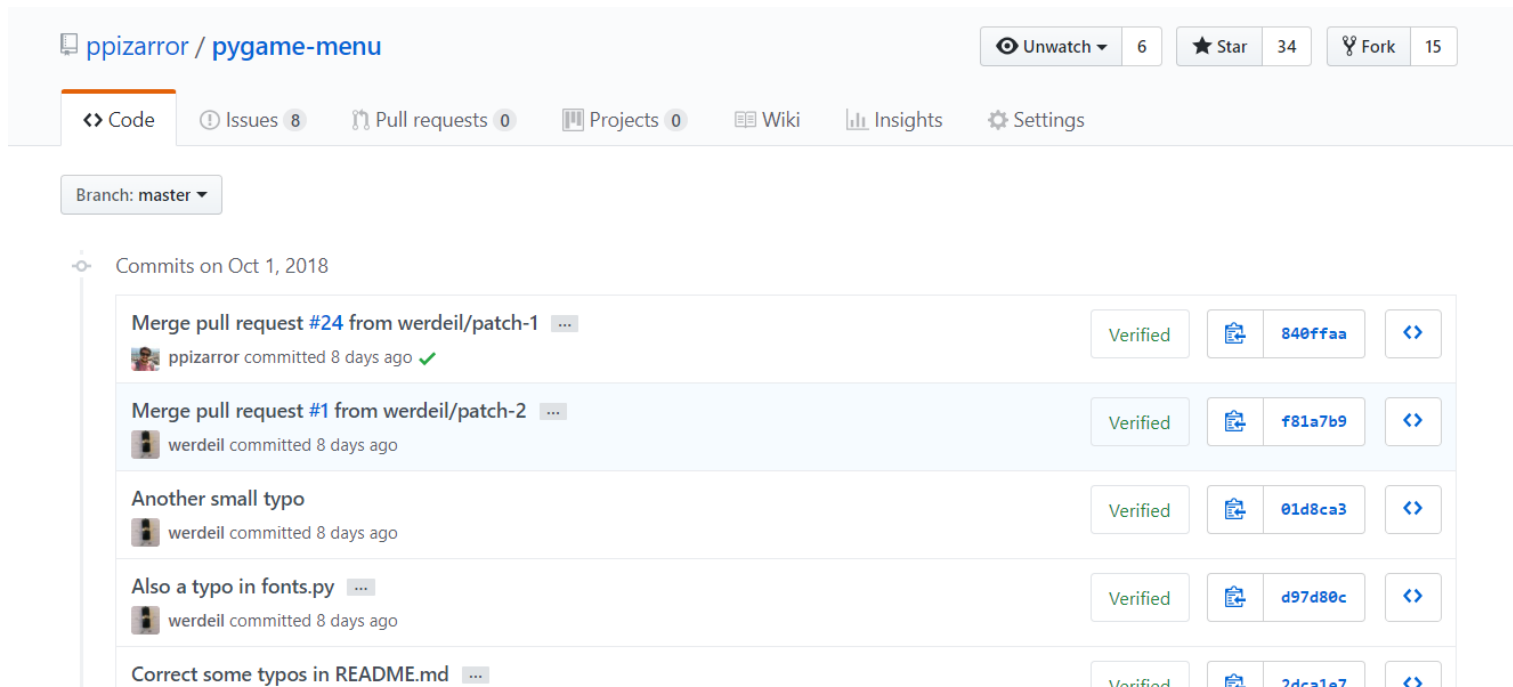
```

On the left, a file explorer shows the repository structure:

- Nombre
- pygameMenu
- .gitignore
- _config
- a
- b
- example
- example2
- LICENSE
- MANIFEST.in
- readme.content

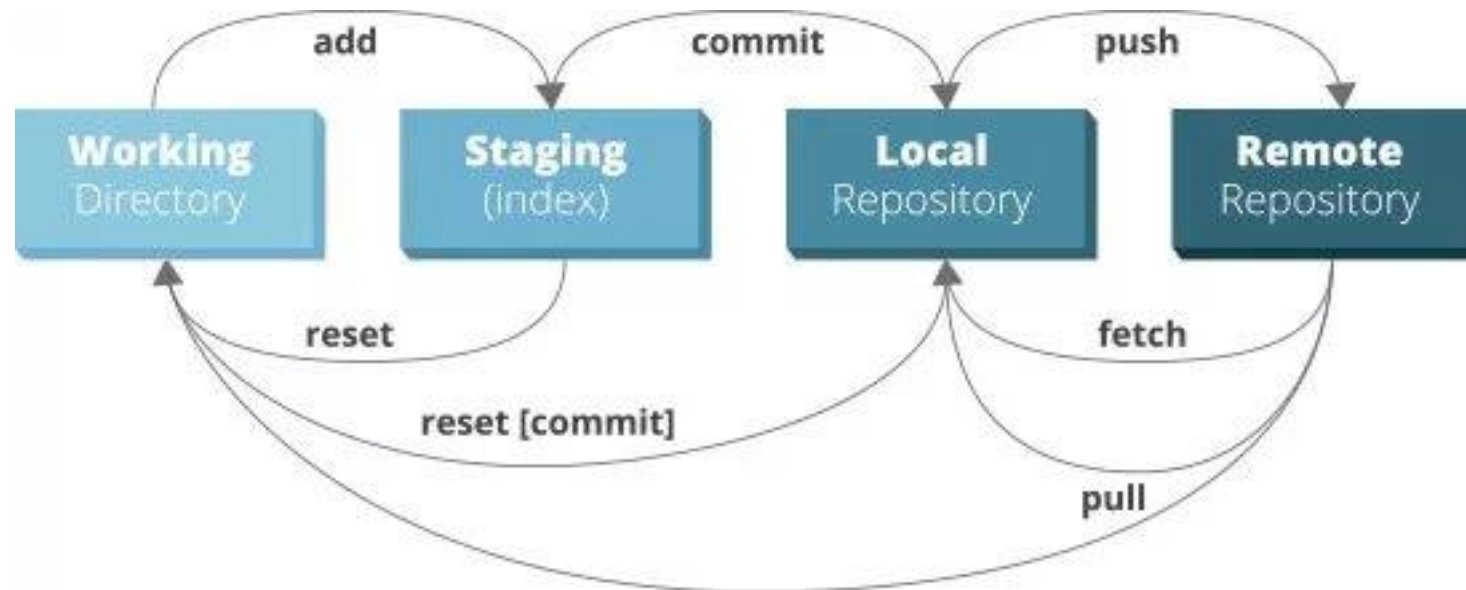
Git – Publicar los commits

- Para publicar los commits al repositorio maestro se debe utilizar `git push`, literalmente se empujan los nuevos cambios.
- En el repositorio se puede revisar todos los commits realizados.



Git – Revisar los commits

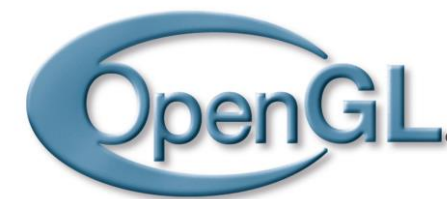
- Es posible descargar el proyecto en una fecha anterior a cada commit.
- Se puede observar cada cambios de los commits.
- Para descargar los commits a tu computador utilizar el comando `git pull`



Git – Descargar los commits

- Al realizar git pull es posible que se puedan generar conflictos (se modifica un archivo que ustedes también modificaron), para ello cada caso se debe revisar de manera manual.
- Cada archivo *conflictivo* se debe luego añadir a un nuevo commit con `git add <filename>`
- Eso sólo ocurre cuando se trabaja en un proyecto de varios colaboradores

OpenGL



- API multilenguaje y multiplataforma para aplicaciones en 2D y 3D.
- Cuenta con más de 250 funciones diferentes.



Shaders

- Shaders son programas personalizados que ejecutan partes del pipeline de rendering gráfico.
- Se ejecutan en la GPU.
- Se utilizarán 2 tipos de shaders:
 - Vertex Shader
 - Fragment Shader

Vertex Shader

- Especificaciones del objeto sobre los vértices

```
vertex_shader = ""  
    #version 130  
    in vec3 position;  
    in vec3 color;  
  
    out vec3 fragColor;  
  
    void main()  
    {  
        fragColor = color;  
        gl_Position = vec4(position, 1.0f);  
    }  
    ""
```

Fragment Shader

- Control sobre los píxeles con la información de los vértices.

```
fragment_shader = ""  
#version 130  
  
in vec3 fragColor;  
out vec4 outColor;  
  
void main()  
{  
outColor = vec4(fragColor, 1.0f);  
}  
""
```

Crear un Shader Program

```
# Assembling the shader program (pipeline) with both shaders
shaderProgram = OpenGL.GL.shaders.compileProgram(
    OpenGL.GL.shaders.compileShader(vertex_shader, GL_VERTEX_SHADER),
    OpenGL.GL.shaders.compileShader(fragment_shader, GL_FRAGMENT_SHADER))

# Telling OpenGL to use our shader program
glUseProgram(shaderProgram)
```

Actividades:

- Clonar repositorio de auxiliar

https://ddonosoc@bitbucket.org/ddonosoc/auxiliar_2.git

Actividades:

1. Dibuje punto a punto una pirámide, utilice un color claro para la cara que da al sol, y una tonalidad más oscura para una cara inclinada levemente visible.
2. Modificar fragment shader como en 5) de actividad (o en Readme del proyecto).
3. Configure dos shader programs. Uno para visualizar de día y otro para la noche. Al presionar la tecla `KEY_ENTER` se debe alternar el shader program que se esté utilizando (cambiando la imagen entre día y noche));

Actividad sugerida

- Agregar dos pirámides.