

# **Administración de Sistemas Operativos**

Práctica Virus & Git

**Marcos Rodriguez Black**

Ciclo Formativo Grado Superior  
Administración de Sistemas Informáticos en Red  
2024-2025

## Índice

Práctica Virus & Git.....	1
1. Descripción de la Tarea.....	3
2. Repaso de Git.....	6
3. Creación del Repositorio en GitHub.....	8
4. Commit Inicial.....	9
5. Implementación Incremental.....	13
Webografia.....	16

# 1. Descripción de la Tarea

Vamos a aprender a usar Git para desarrollar proyectos de software. Lo haremos creando un script bash en la máquina local para la cual haremos un repositorio de git local al que iremos añadiendo commits.

Formalmente se nos ha suministrado un enunciado a resolver con cuatro partes a completar y una lista de requisitos.

El ejercicio seguirá unos principios de hacking ético que haremos patentes en el desarrollo del script mediante las elecciones de diseño y de manera explícita en avisos incluidos en el proyecto.

## 1.1. Enunciado

### Título

Desarrollo de un "Virus" Informático Simulado con Bash/Batch y Control de Versiones en GitHub.

### Objetivo

Aprender a utilizar Git y GitHub para gestionar un proyecto incremental, aplicando buenas prácticas de control de versiones mientras se exploran conceptos básicos de scripting en Bash (Linux/macOS) o Batch (Windows). Nota: El "virus" creado será inofensivo y solo simulará comportamientos para fines educativos.

### Descripción del Ejercicio

#### 1. Creación del Repositorio en GitHub:

- Crea un repositorio público en GitHub llamado *ejercicioGitmrB*.
- Incluye un archivo **README.md** con una descripción del proyecto y una **advertencia clara** de que el código es para fines educativos y no debe ejecutarse en entornos reales sin supervisión.

#### 2. Commit Inicial:

- Clona el repositorio en tu máquina local.
- Crea un script (**virus.sh** para Bash o **virus.bat** para Batch) que muestre un mensaje humorístico o asustadizo (ej: "¡Tu sistema está infectado! 😈").
- Sube este commit con el mensaje: **"Inicio del proyecto: mensaje simulado"**.

### 3. Implementación de "Funcionalidades" Incrementales:

Realiza al menos **4 commits adicionales**, cada uno agregando una característica simulada al script. Ejemplos:

- **Payload Simulado:** Un bucle que "inunde" un archivo temporal con texto aleatorio (ej: `touch /tmp/infectado.txt` y escribir en él).
- **Propagación Simulada:** Copiar el script a otra carpeta (ej: `Documents/Infectados`).
- **Ofuscación:** Codificar parte del script en Base64 y decodificarlo al ejecutar.
- **Autodestrucción:** Eliminar el script después de ejecutarse (ej: `rm virus.sh` o del `virus.bat`).

*Nota: Asegúrate de que toda las acciones sean reversibles y no dañen el sistema real.*

### 4. Commit Final y Ética:

- Agrega un archivo `DISCLAIMER.txt` enfatizando que el proyecto es educativo y no debe usarse maliciosamente.
- Actualiza el `README.md` explicando el propósito académico del repositorio.

## Requisitos de la entrega

- Enlace al repositorio de GitHub con al menos 20 commits significativos (incluyendo el inicial).
- Cada commit debe tener un mensaje descriptivo (ej: "**FEAT:** `agregada propagación simulada`").
- El script debe incluir comentarios explicando cada sección,

## Criterios de Evaluación:

- Correcto uso de Git/GitHub (ramas, mensajes, estructura).
- Originalidad y creatividad en las funcionalidades simuladas.
- Claridad en las advertencias éticas incluidas.
- *Penalización si el código incluye acciones reales dañinas.*

Con la primera entrega se hará una revisión del código y el profesor hará propuestas que se deberán implementar en un branch.

Entregar la url del repositorio de Github **PUBLICO**.

## 1.2. Solución propuesta

Crearé un script llamado `gnomed.bash` que periódicamente muestre en terminal un arte ASCII de temática gnómica.

### Funcionamiento del Script

En la primera ejecución el script se copiará a un directorio oculto en el `$HOME` del usuario que lo lance y borrará el fichero original. Se añadirá al crontab del usuario para ejecutarse periódicamente. Elegirá un arte aleatorio de entre una serie de ficheros de texto ASCII que imprimirá por la salida estándar de terminal. [Enlace](#) a un ejemplo de posibles salidas.

### Entorno de prueba

El script será diseñado para un sistema linux usado en terminal, las pruebas las realizaré en una máquina virtual Ubuntu 22.04 LTS.

### Software utilizado

Además de git se ha usado el editor de texto [Vim](#) para el desarrollo del software, [Ubuntu Server](#) para la máquina virtual de pruebas, [VirtualBox](#) para la emulación y [Libreoffice Writer](#) para elaborar el presente documento.

## 2. Repaso de Git

Los primeros capítulos del [libro de Git](#) ayudan a entender el software mejor que cualquier tutorial.

### ¿Qué es?

Git es un software de control de versiones (VCS), que a diferencia de los anteriores a éste no guarda la información una lista de cambios en ficheros. En lugar de conceptualizar los datos como un conjunto de archivos y cambios en el tiempo (Conocido como método delta de control de versión) Git piensa en proyecto como una instantánea de un pequeño sistema de archivos.

Un fichero son cambios no se guarda en la siguiente instantánea, se referencia. Git no requiere conexión al repositorio central para operar en el proyecto, lo hace sobre una copia local que más adelante se combinará con el repositorio principal.

Todo elemento en Git se referencia por su hash, de modo que no es posible hacer alteraciones al proyecto sin que Git lo sepa. Volver a una versión anterior consiste en calcular diferencias en un sistema de archivos local, de modo que el usuario no se carga el proyecto accidentalmente al ir cambiando de una versión a otra.

### Conceptos clave

Git considera que un fichero puede estar en tres estados:

- **Modified** – Se ha alterado el fichero pero no está en la base de datos.
- **Staged** – El fichero modificado se incluirá en la próxima instantánea.
- **Committed** – El fichero está guardado en la base de datos.

Esto nos lleva a tres secciones de un proyecto Git: Working Directory, Staging area y .git directory (repository).

Git guarda la información *Staged* en un fichero local llamado *Index*.

## Configuración de Git

En orden descendente los siguientes ficheros sobrescriben la configuración de Git

1. `[path]/etc/gitconfig` - Valores para todo usuario y todo repositorio. Accesible mediante `git config --system`.
2. `~/.gitconfig` - Tu usuario, todo repositorio. `git config --global`.
3. `.git/config` - Un repositorio. `git config --local`.

Se puede consultar su estado y origen con `git config --list --show-origin`. Lo primero que uno debiera hacer para trabajar con Git es establecer su nombre y email:

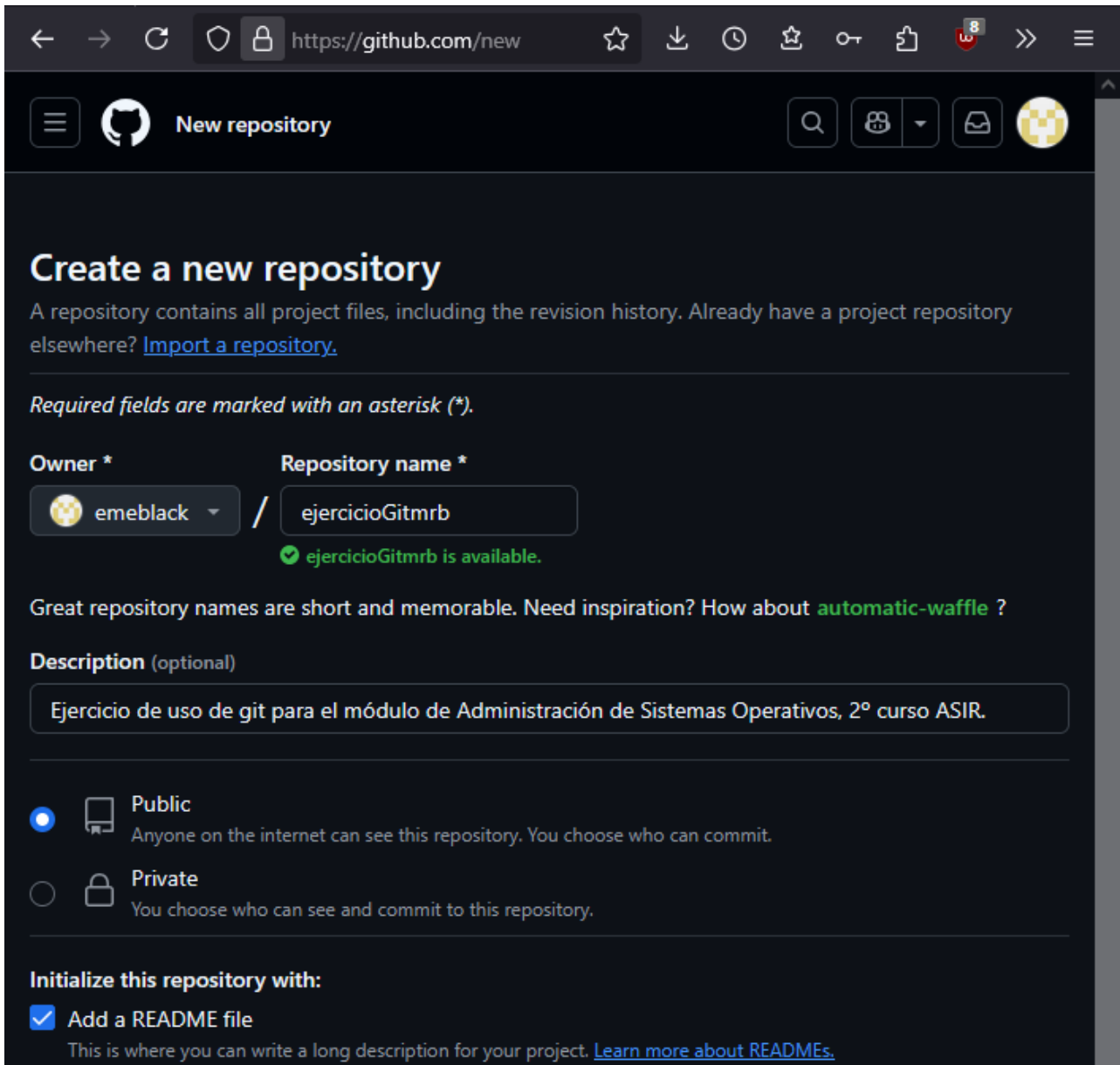
```
git config --global user.name "emeblack"
git config --global user.email mrodbla937@ieszaidinvergeles.org
```

También es útil indicar un [editor de texto](#) por defecto para Git por ejemplo:

```
git config --global core.editor \
"C:\Program Files\Vim\vim72\gvim.exe" --nofork '%*'
```

### 3. Creación del Repositorio en GitHub

Navegamos a Github y creamos una repo nueva.



← → ↻ 🔒 https://github.com/new ☆ ⬇ ⌚ ⭐ 🔑 📁 8 >> ≡

≡ 🐙 New repository 🔍 🧑 👤 📁 🏠

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

**Owner \*** **Repository name \***

🏠 emeblack / ejercicioGitmrB

✔ ejercicioGitmrB is available.

Great repository names are short and memorable. Need inspiration? How about **automatic-waffle** ?

**Description** (optional)

Ejercicio de uso de git para el módulo de Administración de Sistemas Operativos, 2º curso ASIR.

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Figura 1: Creación de repositorio en Github.

Modificaremos el fichero **README.md** para que muestre una **advertencia clara**. Incluiremos el enunciado del ejercicio hacer patente el propósito completo de la tarea.



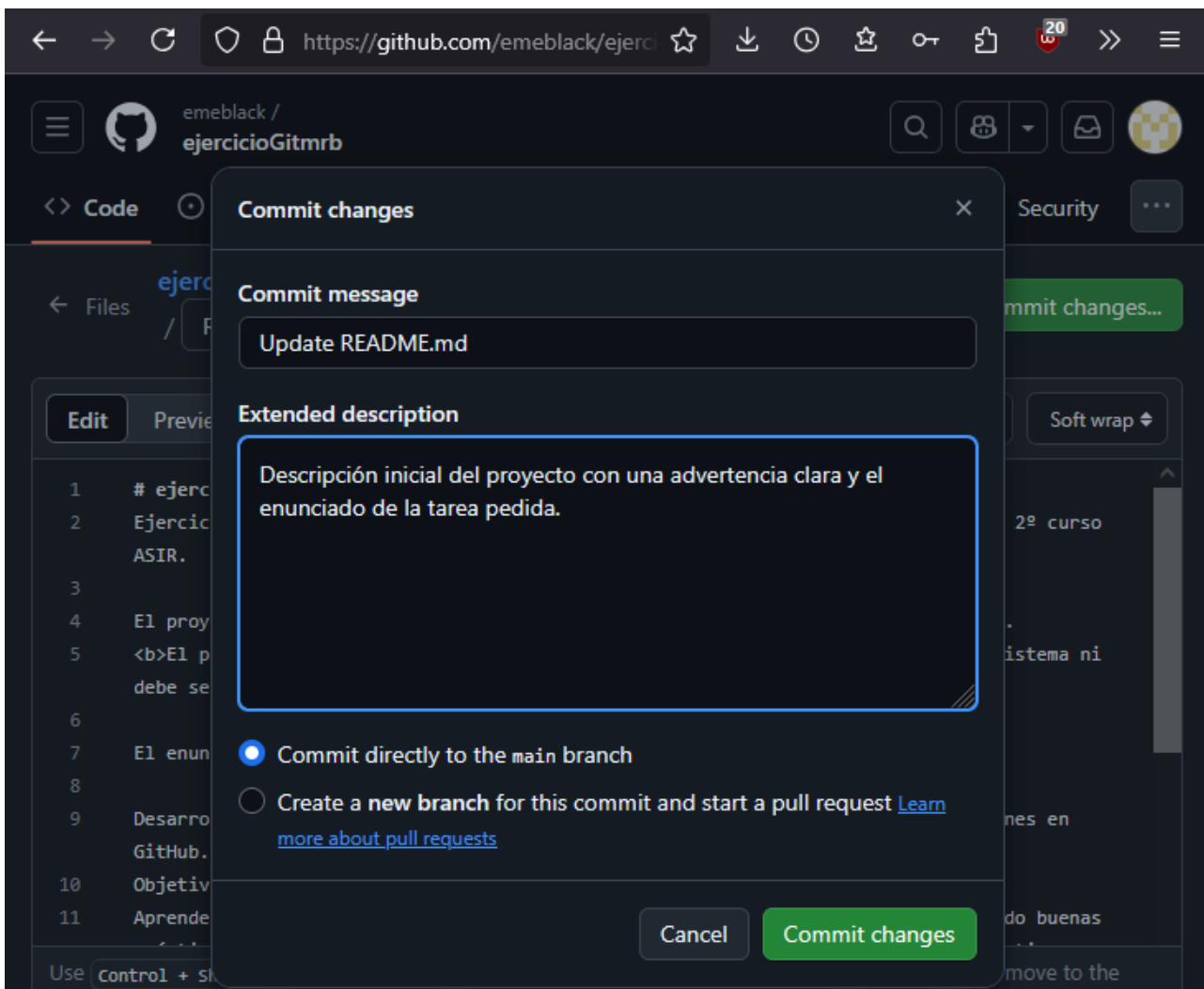


Figura 2: Primer commit actualizando readme.md.

## 4. Commit Inicial

Técnicamente no va a ser el primero – hemos actualizado el readme.md antes – pero sí el primero con desarrollo propiamente dicho.

Primero necesitaremos tener instalado Git en nuestra máquina local, en mi caso trabajaré en una terminal de Windows. [Enlace](#) al tutorial de instalación seguido. Para mostrar que nos hallamos en un repositorio Git local he usado un [prompt personalizado](#).

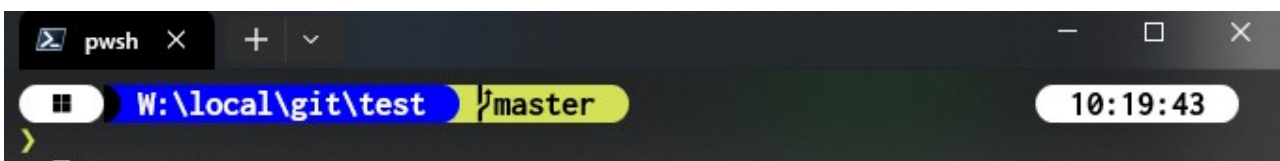
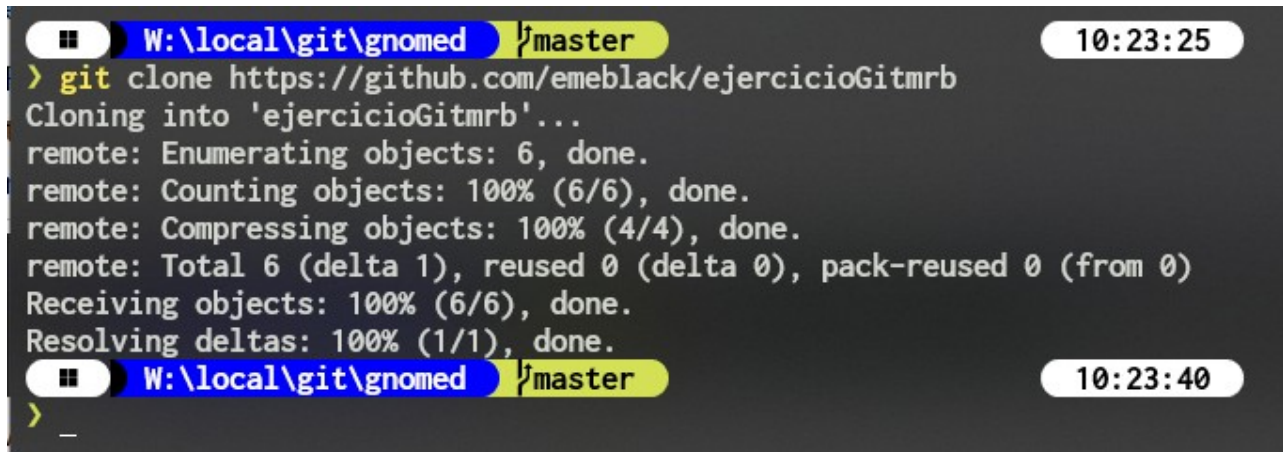


Figura 3: Prompt personalizado Oh My Posh

## 4.1. Clonación

Usamos `git clone` seguido del nombre del repositorio.

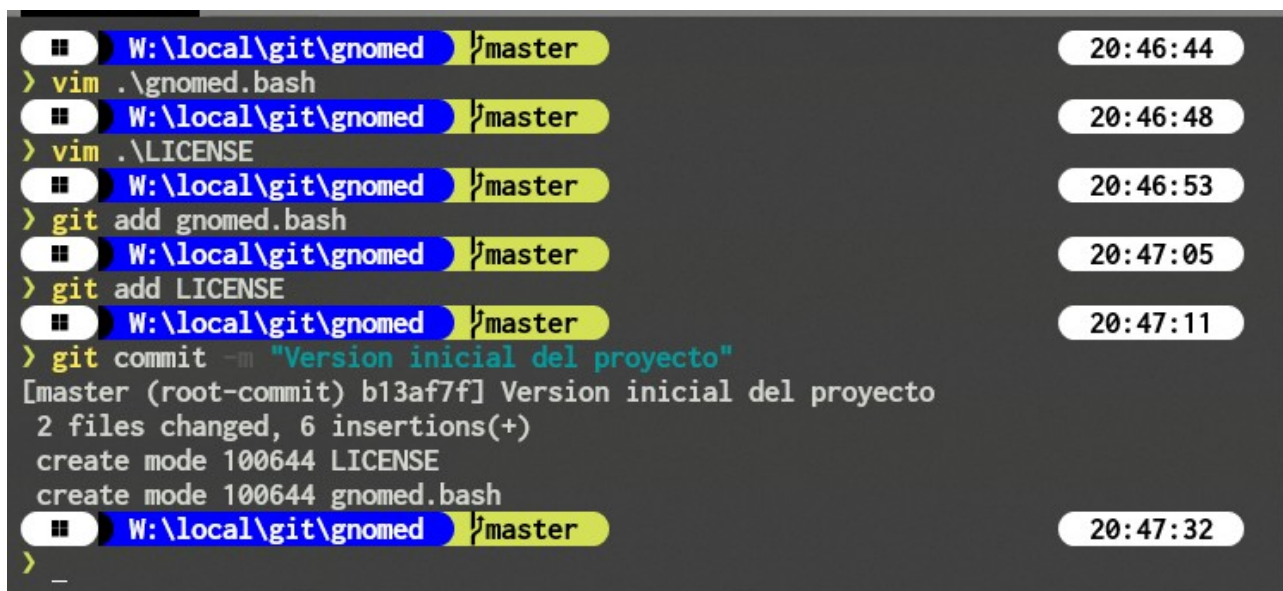


```
W:\local\git\gnomed master 10:23:25
> git clone https://github.com/emeblack/ejercicioGitmr
Cloning into 'ejercicioGitmr'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
Resolving deltas: 100% (1/1), done.
W:\local\git\gnomed master 10:23:40
> _
```

Figura 4: Clonación del repositorio.

## 4.2. Creación de un script

Guardamos un script bash sencillo y una licencia. He optado por [CC BY-SA](#). Los añadimos al commit siguiente:



```
W:\local\git\gnomed master 20:46:44
> vim .\gnomed.bash
W:\local\git\gnomed master 20:46:48
> vim .\LICENSE
W:\local\git\gnomed master 20:46:53
> git add gnomed.bash
W:\local\git\gnomed master 20:47:05
> git add LICENSE
W:\local\git\gnomed master 20:47:11
> git commit -m "Version inicial del proyecto"
[master (root-commit) b13af7f] Version inicial del proyecto
2 files changed, 6 insertions(+)
create mode 100644 LICENSE
create mode 100644 gnomed.bash
W:\local\git\gnomed master 20:47:32
> _
```

Figure 5: Añadimos ficheros al primer commit.

Aún tenemos que subirlo al repositorio. Añadimos el repositorio a la lista de orígenes y hacemos un push:

```
git remote add origin https://github.com/emeblack/ejercicioGitmrB
```

### 4.3. Primer Push

Tendremos que introducir nuestras credenciales de github para continuar.

```
W:\local\git\gnomed master 20:56:48
> git push origin master
info: please complete authentication in your browser...
```

Figura 6: Git nos pide credenciales para hacer el push.

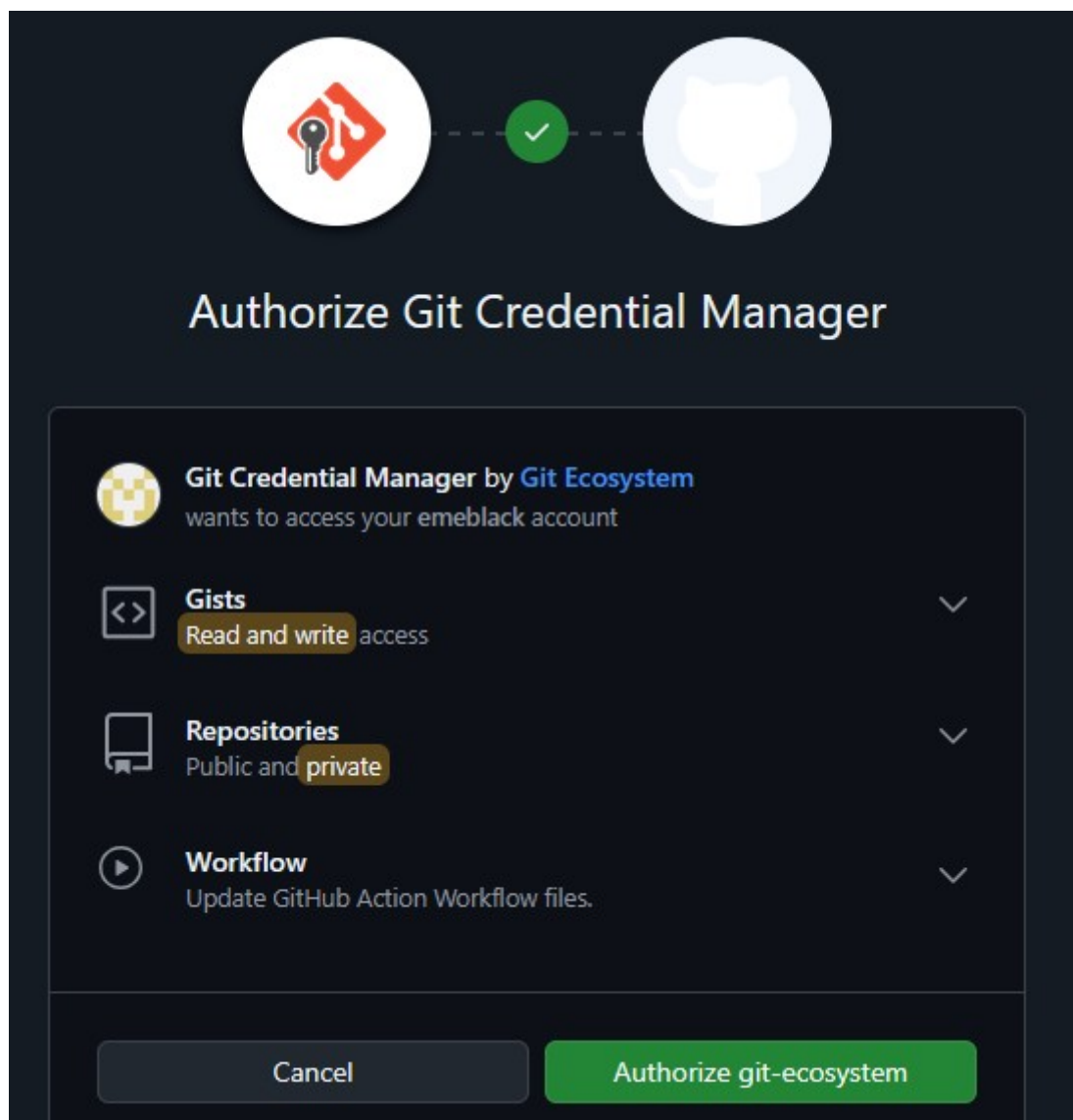


Figura 7: Autorización de credenciales en github.com.

Y de vuelta a la terminal vemos que el proceso se ha ejecutado correctamente.

```
W:\local\git\gnomed master 20:56:48
> git push origin master
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 392 bytes | 392.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/emeblack/ejercicioGitmr/pull/new/master
remote:
To https://github.com/emeblack/ejercicioGitmr
 * [new branch]   master -> master
W:\local\git\gnomed master 21:00:02
>
```

Figura 8: uso de git push.

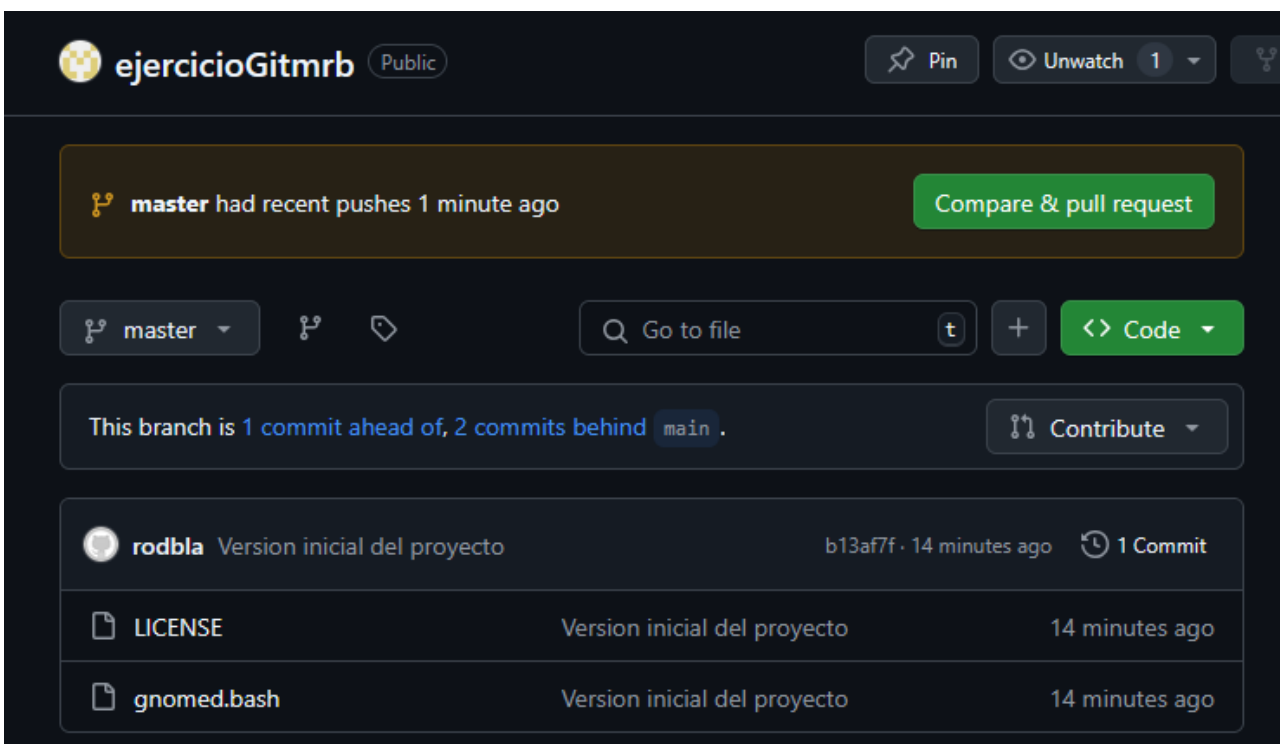


Figura 9: Cambios vistos en al web.



## 5. Implementación Incremental

Vamos a añadir los 4 commits iniciales. Tras cada commit haremos un commit y un push para subirlos a github.

### 5.1. Payload simulado

crearemos un fichero `$HOME/infectado.txt` con 512 bytes de contenido aleatorio. Clonamos el repositorio en una máquina virtual de prueba y comprobamos el resultado:

```
dd if=/dev/urandom of=$HOME/infectado.txt bs=512 count=1
```

```
usuario@ubusrv:~$ git clone -b master https://github.com/emeblack/ejercicioGitmrB/
Cloning into 'ejercicioGitmrB'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 13 (delta 2), reused 6 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (2/2), done.
usuario@ubusrv:~$ cd ejercicioGitmrB/
usuario@ubusrv:~/ejercicioGitmrB$ ls
gnomed.bash  LICENSE
usuario@ubusrv:~/ejercicioGitmrB$ chmod u+x gnomed.bash
usuario@ubusrv:~/ejercicioGitmrB$ ./gnomed.bash
¡Tu sistema está infectado! 🐼
1+0 records in
1+0 records out
512 bytes copied, 0,000693405 s, 738 kB/s
usuario@ubusrv:~/ejercicioGitmrB$ cd ..
usuario@ubusrv:~$ ls
ejercicioGitmrB  infectado.txt  netplan
usuario@ubusrv:~$
```

Figura 10: Prueba del payload.

Nótese que aún no hemos fusionado la rama 'master' con 'main', hemos de especificar qué rama queremos clonar con la opción `-b`.



## 5.5. Ejecución automática

Añadimos al crontab del usuario el script. No imprimirá el mensaje del gnomon en terminal pues cron funciona sin una terminal a la que imprimir, pero hará el resto de funciones del script.

```
(crontab -l 2>/dev/null; echo "* * * * * /home/usuario/test.bash") \  
| crontab -
```

## 5.6. Mejoras propuestas

Lo suyo sería hacer que el script se ejecutara solo sin recurrir a crontab para poder hacer uso de la terminal de usuario. Una alternativa sería usar una unidad de systemd a nombre del usuario y asignarle tty1, pero elaborar ésta construcción en un script escapa al propósito de ésta práctica.

## 5.7 Commit final

Solo nos queda añadir al proyecto el fichero **DISCLAIMER.txt** indicando el propósito educativo del proyecto:

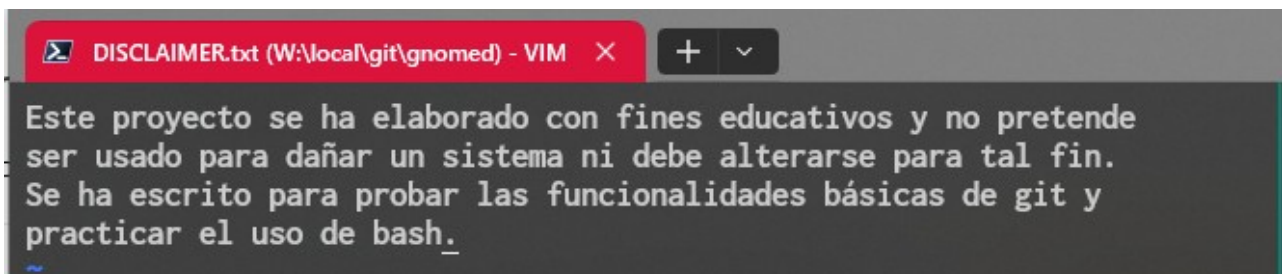


Figura 12: DISCLAIMER.txt

Y hacemos un último commit del proyecto junto con un push:

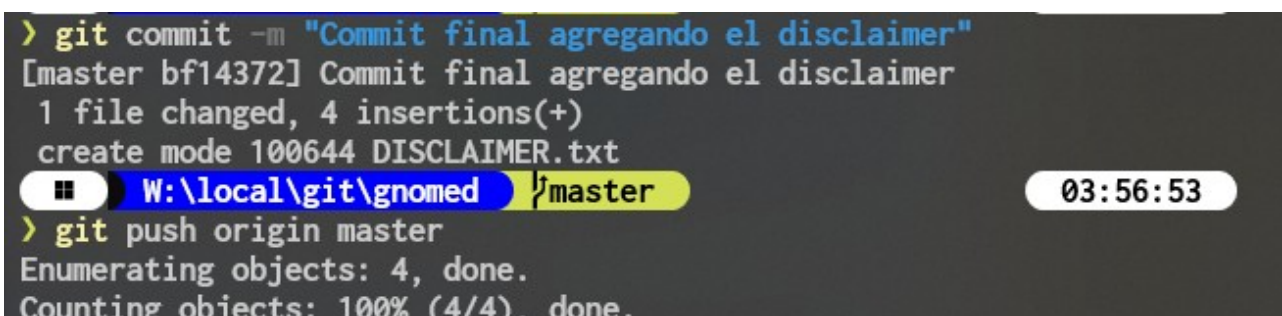


Figura 13: commit final del proyecto.

## Webografia

- [github emeblack – ejercicioGitmrh](#)
- [git – book](#)
- [simplilearn – Install Git Windows](#)
- [creative commons – CC BY-SA 4.0](#)
- [stackoverflow – Cronjob with non-root user](#)
- [superuser – Create a file of random bytes quickly](#)
- [askubuntu – decode base64 on command line](#)
- [stackoverflow – crontab through a script](#)