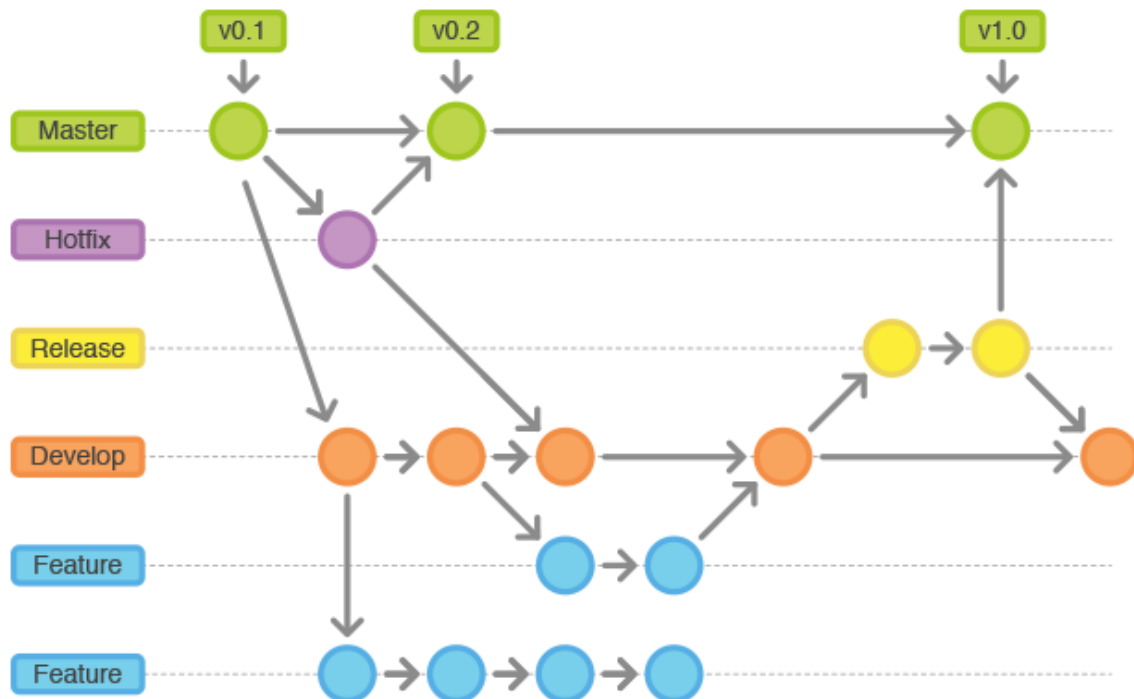


8 DE MAYO DE 2017



CONTROL DE VERSIONES

GIT Y HOSTING DE REPOSITORIOS

EMILIANO MONTESDEOCA DEL PUERTO

CIFP CESAR MANRIQUE
ENTORNOS DE PROGRAMACION

Índice

1. Enunciado.
2. Objetivo de la práctica.
3. Introducción a Git y control de versiones.
4. Uso de ramas
5. Github, BitBucket, GitLab y más gestores de control de versiones.
6. Ejercicios de la práctica.
 - a. Creamos una carpeta para el proyecto: Proyecto1
 - b. En esta carpeta creamos 2 subcarpetas: Código1 y Código2
 - c. Creamos archivos texto con el Wordpad en las carpetas.
 - d. En Proyecto1: uno.txt y dos.txt
 - e. En Código1: tres.txt y cuatro.dat
 - f. En Código2: cinco.txt
 - g. Iniciamos la carpeta Proyecto1 como repositorio GIT
 - h. Configuramos nuestro usuario y correo
 - i. En el archivo exclude, que está en la carpeta oculta .git, excluimos los archivos con extensión *.dat
 - j. Ahora preparamos los archivos para llevarlos al Stage
 - k. Comprobamos lo realizado con git status
 - l. Confirmamos lo que hay en el Stage al Repositorio
 - m. Modificamos los archivos uno.txt y cinco.txt
 - n. Comprobamos de nuevo el estado
 - o. Ver las diferencias entre los archivos con git diff
 - p. Preparamos los archivos cambiados
 - q. Confirmamos al Repositorio.
 - r. Ver mediante log los commits realizados.
 - s. Restauramos los archivos uno.txt y cinco.txt de uno en uno al Working Directory
 - t. Comprobamos de nuevo el estado.
 - u. Crear el repositorio nuevo en GitHub
 - v. Crear el enlace desde GIT al Repositorio remoto en GitHub
 - w. Ver los enlaces que tenemos
 - x. Subir a GitHub el repositorio local
 - y. Comprobar en GitHub la presencia del repositorio con los ficheros subidos.
7. Bibliografía

Enunciado

Se pide:

1. Creamos una carpeta para el proyecto: Proyecto1
2. En esta carpeta creamos 2 subcarpetas:Codigo1 y Codigo2
3. Creamos archivos texto con el Wordpad en las carpetas.
4. En Proyecto1: uno.txt y dos.txt
5. En Codigo1: tres.txt y cuatro.dat
6. En Codigo2: cinco.txt
7. Iniciamos la carpeta Proyecto1 como repositorio GIT
8. Configuramos nuestro usuario y correo
9. En el archivo exclude, que está en la carpeta oculta .git, excluimos los archivos con extensión *.dat
10. Ahora preparamos los archivos para llevarlos al Stage
11. Comprobamos lo realizado con git status
12. Confirmamos lo que hay en el Stage al Repositorio
13. Modificamos los archivos uno.txt y cinco.txt
14. Comprobamos de nuevo el estado
15. Ver las diferencias entre los archivos con git diff
16. Preparamos los archivos cambiados
17. Confirmamos al Repositorio.
18. Ver mediante log los commits realizados.
19. Restauramos los archivos uno.txt y cinco.txt de uno en uno al Working Directory
20. Comprobamos de nuevo el estado.
21. Crear el repositorio nuevo en GitHub
22. Crear el enlace desde GIT al Repositorio remoto en GitHub
23. Ver los enlaces que tenemos
24. Subir a GitHub el repositorio local
25. Comprobar en GitHub la presencia del repositorio con los ficheros subidos.

Seguir esta secuencia e ir en cada paso poniendo en el informe lo realizado, los resultados y la explicación. Vayan capturando pantallas de todo el proceso.

Recuerden que en el informe deben de estar detallados los comandos ejecutados, sus resultados y su explicación.

Incluir como siempre el Índice, Enunciado y Objetivos.

Módulo de Entornos de Desarrollo 1º DAW.

Jorge Rivero, profesor del módulo.

Santa Cruz de Tenerife a 4 de abril de 2017

Objetivo de la practica

El objetivo de esta práctica es enseñar el uso de software de control de versiones, donde nos da ventajas a la hora de crear software, ya que nos permite mantener una copia de seguridad de nuestro código por versiones, en caso de que un cambio haga que el código no compile.

El uso de control de versiones actualmente estrictamente necesario, sobre todo, en equipos, ya que muchas personas trabajando en un solo proyecto con mucho código puede generar conflictos, y para poder evitar eso, se puede utilizar herramientas incluidas en **Git** para revertir los cambios y volver a utilizar una versión que funcione correctamente.

Introducción a Git y control de versiones.



Git es un software de control de versiones diseñado por **Linus Torvalds**, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Al principio, **Git** se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, **Git** se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena.

Hay algunos proyectos de mucha relevancia que ya usan **Git**, en particular, el grupo de programación del núcleo Linux.

Control de versiones

¿Qué control de versiones? Pues bien, se define como control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo es decir a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración, y para los que aún no les queda claro del todo, control de versiones es lo que se hace al momento de estar desarrollando un software o una página web.

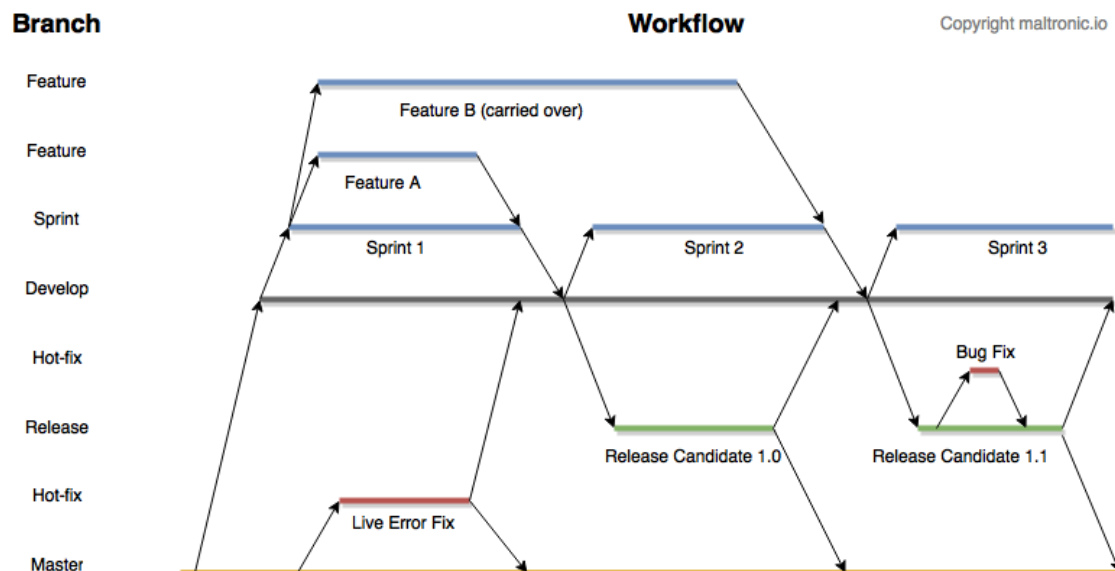
Exactamente es eso que haces cuando subes y actualizas tu código en la nube, o le añades alguna parte o simplemente le editas cosas que no funcionan como deberían o al menos no como tú esperarías.

Uso de ramas

Las ramas, en un sistema de control de versiones, constituyen una potente herramienta que flexibiliza la forma en la que los colaboradores cooperan en el proyecto (en inglés Branching Workflows). Las ramas son solo una herramienta que es posible utilizar de distintas formas para conseguir distintos objetivos, hay varios tipos, pero los más importantes son:

- **Ramas de largo recorrido**
 - a. En algunos proyectos se tienen varias ramas siempre abiertas, que indican diversos grados de estabilidad del contenido. Por ejemplo, en la rama 'master' es frecuente mantener únicamente lo que es totalmente estable. Luego se tienen otras ramas que revelan distintos grados de estabilidad. Por ejemplo podríamos tener una rama 'beta' (versión beta) y otra 'alfa' (versión alfa), en las que se va trabajando. Cuando se alcanza cierto grado de estabilidad superior a la rama en la que se está entonces se realiza una fusión con rama de estabilidad superior.
- **Ramas puntuales**
 - b. Las ramas puntuales, también llamadas ramas de soporte, son ramas que se crean de forma puntual para realizar una funcionalidad muy concreta. Por ejemplo añadir una nueva característica (se les llama ramas de nueva funcionalidad o directamente por su nombre en inglés topic branch ó feature branch) o corregir un fallo concreto (se les llama ramas para corregir error o directamente por su nombre en inglés hotfix branch).

Aquí tenemos un ejemplo de un proyecto con ramas, donde diferentes desarrolladores trabajan al mismo tiempo.



Como podemos ver, la master tiene diferentes **ramas de largo recorrido** (develop, hot-fix, reléase) y luego **ramas puntuales** (feature, sprint).

Github, BitBucket, GitLab y más gestores de control de versiones.

Los gestores de repositorios son empresas que dejan servidores para que los usuarios puedan subir sus repositorios. Estas empresas tienen aplicaciones que permiten ver el control de versiones de formas más ordenada, aparte de poder compartir repositorios con otros usuarios y así poder trabajar en equipo. Dentro de los destacados, el más famoso actualmente es **GitHub**, **BitBucket**, **Gitlab**, pero hay bastantes más.

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc.

(anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

The GitHub logo, featuring the word "GitHub" in a bold, black, sans-serif font.

Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Bitbucket ofrece planes comerciales y gratuitos.

Se ofrece cuentas gratuitas con un número ilimitado de repositorios privados (que puede tener hasta cinco usuarios en el caso de cuentas gratuitas) desde septiembre de 2010, los repositorios privados no se muestran en las páginas de perfil - si un usuario sólo tiene depósitos privados, el sitio web dará el mensaje "Este usuario no tiene repositorios". El servicio está escrito en Python.



GitLab es un proyecto de software libre de la compañía del mismo nombre que está programado en Ruby que, quizá siendo un poco simplista, podríamos decir que es simplemente un frontend de Git. No es solo un frontend web de Git

y, por este nombre, lo primero que nos puede venir a la cabeza son estos antiguos frontends web que había, que solo nos permitían ver los ficheros y sus últimos cambios. Sin embargo, GitLab es una suite completa que nos permite gestionar, administrar, crear y conectar nuestros repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas. No solo incluye estos módulos para revisar los ficheros, sino que, además, con facilidad, podemos revisar diffs, de una manera muy visual, de todos nuestros commits, y ver dónde se producen los cambios.

The word "GitLab" in a grey, sans-serif font.

Aparte de estos gestores de repositorios, existen más utilizados, como CodePlex, LaunchPad, etc.

1-6. Generar estructura

- a. Creamos una carpeta para el proyecto: Proyecto1
- b. En esta carpeta creamos 2 subcarpetas: Código1 y Código2
- c. Creamos archivos texto con el Wordpad en las carpetas.
- d. En Proyecto1: uno.txt y dos.txt
- e. En Código1: tres.txt y cuatro.dat
- f. En Código2: cinco.txt

```
C:\Users\Emi\Desktop>tree Proyecto1 /F
Listado de rutas de carpetas
El número de serie del volumen es 00000253 FA27:8871
C:\USERS\EMI\DESKTOP\PROYECTO1
├── dos.txt
├── uno.txt
├── Código1
│   ├── cuatro.dat
│   └── tres.txt
└── Código2
    └── cinco.txt
```

8. Iniciamos la carpeta Proyecto1 como repositorio GIT

Para iniciar la carpeta como repositorio hay que utilizar el comando *git init*.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1
$ git init
Initialized empty Git repository in C:/Users/Emi/Desktop/Proyecto1/.git/
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

Esto genera una carpeta oculta .git que contiene archivos para el repositorio

Este equipo > Escritorio > Proyecto1		
Nombre		Fecha de i
	.git	05/05/201
	Código1	26/04/201
	Código2	26/04/201
	dos.txt	26/04/201
	uno.txt	26/04/201

9. Configuramos nuestro usuario y correo

Para configurar el usuario y correo, accedemos a la configuración global del repositorio con el comando **config --global**, y ahí podemos cambiar el **user.name** y el **user.email**

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git config --global user.name Emi

Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git config --global user.email emontesdeoc@gmail.com
```

10. En el archivo **exclude**, que está en la carpeta oculta **.git**, excluimos los archivos con extensión ***.dat**

El archivo **exclude** se incluyen los archivos que no se quieren subir cuando se hace un **commit**.

```
PS C:\Users\Emi\Desktop\Proyecto1\.git> tree /f
Listado de rutas de carpetas
El número de serie del volumen es 0000008E FA27:8871
C:..
|   config
|   description
|   HEAD
|---hooks
|       applypatch-msg.sample
|       commit-msg.sample
|       post-update.sample
|       pre-applypatch.sample
|       pre-commit.sample
|       pre-push.sample
|       pre-rebase.sample
|       pre-receive.sample
|       prepare-commit-msg.sample
|       update.sample
|---info
|       exclude
|---objects
|       info
|       pack
```

```
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good set of
# exclude patterns (uncomment them if you want to use them):
# *.o
# *~
*.dat
```


11. Ahora preparamos los archivos para llevarlos al Stage.

Para ver el estado del repositorio y sus archivos, se utiliza el comando **git status**, que muestra si hay archivos no puestos para subir, se ven de color rojo.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git status
On branch master

Initial commit

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

       Codigo1/
        Codigo2/
        dos.txt
        uno.txt

nothing added to commit but untracked files present (use "git add" to track)
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

Para subirlos se pueden realizar varios comandos, el que utilizo yo es **git add --all**

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git add --all
```

12. Comprobamos lo realizado con git status

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git status
On branch master

Initial commit

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

        new file:   Codigo1/cuatro.dat
        new file:   Codigo1/tres.txt
        new file:   Codigo2/cinco.txt
        new file:   dos.txt
        new file:   uno.txt

Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

Una vez añadido todos los archivos, realizamos un **git status** y vemos que los archivos que antes estaban en rojo ahora están verdes, porque están listos para ser subidos al repositorio remoto

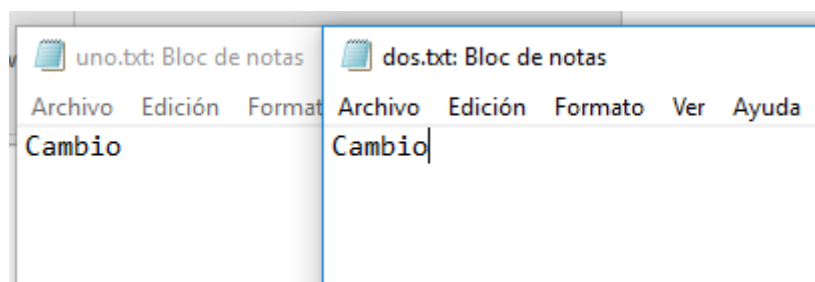
13. Confirmamos lo que hay en el Stage al Repositorio

Para añadir estos archivos al repositorio, utilizamos el comando **git commit -m "X"**, donde X es el título del **commit**.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git commit -m "Confirmacion"
[master (root-commit) 4396dae] Confirmacion
5 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644Codigo1/cuatro.dat
create mode 100644Codigo1/tres.txt
create mode 100644Codigo2/cinco.txt
create mode 100644dos.txt
create mode 100644uno.txt

Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

14. Modificamos los archivos uno.txt y cinco.txt



15. Comprobamos de nuevo el estado

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git status
On branch master
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:   dos.txt
        modified:   uno.txt

no changes added to commit (use "git add" and/or "git commit -a")

Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

Para ver en general si hay algún cambio, hacemos un **git status** y se mostrara si hay algún archivo modificado.

16. Ver las diferencias entre los archivos con git diff

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git diff
diff --git a/dos.txt b/dos.txt
index e69de29..ec216bf 100644
--- a/dos.txt
+++ b/dos.txt
@@ -0,0 +1 @@
+Cambio
\ No newline at end of file
diff --git a/uno.txt b/uno.txt
index e69de29..ec216bf 100644
--- a/uno.txt
+++ b/uno.txt
@@ -0,0 +1 @@
+Cambio
\ No newline at end of file
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

Con el comando **git diff**, nos muestra las diferencias de los archivos modificados.

17. Preparamos los archivos cambiados

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git add --all
```

18. Confirmamos al Repositorio.

Realizamos un **commit** para subir los cambios

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git commit -m "Cambios"
[master a40ccc4] Cambios
2 files changed, 2 insertions(+)
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ |
```

19. Ver mediante log los commits realizados.

Para ver los cambios se utiliza el comando **git log**, que muestra todos los **commits** realizados.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git log
commit a40ccc43334ac6a870db200427af4dfc3d1ae7ed
Author: Emi <emontesdeoc@gmail.com>
Date: Fri May 5 11:56:20 2017 +0100

Cambios

commit 4396dae782b55c520e818af9f3b96b33a4f87c20
Author: Emi <emontesdeoc@gmail.com>
Date: Fri May 5 11:52:48 2017 +0100

Confirmacion
```

20. Restauramos los archivos uno.txt y cinco.txt de uno en uno al Working Directory

Para restaurar un **commit** realizado se tiene que saber el ID del **commit**, y se utiliza con el comando **git checkout**.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 ((4396dae...))
$ git checkout a40ccc43334ac6a870db200427af4dfc3d1ae7ed
Previous HEAD position was 4396dae... Confirmacion
HEAD is now at a40ccc4... Cambios
```

21. Comprobamos de nuevo el estado.

Git status para saber el estado del repositorio.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 ((a40ccc4...))
$ git status
HEAD detached at a40ccc4
nothing to commit, working tree clean
```

22. Crear el repositorio nuevo en GitHub


En la página web de GitHub creamos un repositorio, para poder subir nuestro repositorio a su servicio de hosting.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name

 emimontesdeoca ▾

/ proyecto1 ✓

Great repository names are short and memorable. Need inspiration? How about **effective-octo-doodle**.

Description (optional)

Repositorio para parctica de GIT

☒ Public

Anyone can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾ ⓘ

Create repository

23. Crear el enlace desde GIT al Repositorio remoto en GitHub

Añadimos el repositorio remoto a nuestro repositorio local

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 ((a40ccc4...))
$ git remote add origin https://github.com/emimontesdeoca/proyecto1.git
```

24. Ver los enlaces que tenemos

Usando **git remote -v** podemos ver los enlaces del repositorio.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 ((a40ccc4...))
$ git remote -v
origin https://github.com/emimontesdeoca/proyecto1.git (fetch)
origin https://github.com/emimontesdeoca/proyecto1.git (push)
```

25. Subir a GitHub el repositorio local

Para subir cambios al repositorio remoto hay que utilizar el comando **git push**.

```
Emi@Emi-Laptop MINGW64 ~/Desktop/Proyecto1 (master)
$ git push --set-upstream origin master

Counting objects: 8, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 655 bytes | 0 bytes/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://github.com/emimontesdeoca/proyecto1.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

26. Comprobar en GitHub la presencia del repositorio con los ficheros subidos.

Refrescando la página web en GitHub deberían de verse los nuevos archivos.

The screenshot shows the GitHub interface for the repository 'emimontesdeoca / proyecto1'. At the top, there are navigation links: 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below these, it says 'Repositorio para parctica de GIT' with an 'Edit' button. A summary bar shows '2 commits', '1 branch', '0 releases', and '1 contributor'. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The main content area shows a list of files: 'Codigo1' (Confirmacion, 13 minutes ago), 'Codigo2' (Confirmacion, 13 minutes ago), 'dos.txt' (Cambios, 10 minutes ago), and 'uno.txt' (Cambios, 10 minutes ago). At the bottom, there is a prompt to 'Add a README'.

File	Change	Time
Codigo1	Confirmacion	13 minutes ago
Codigo2	Confirmacion	13 minutes ago
dos.txt	Cambios	10 minutes ago
uno.txt	Cambios	10 minutes ago

Bibliografía

Para realizar esta práctica se ha utilizado información de varias páginas web listadas a continuación:

1. <https://es.wikipedia.org/wiki/Git>
2. https://es.wikipedia.org/wiki/Control_de_versiones
3. <https://es.wikipedia.org/wiki/Bitbucket>
4. <https://es.wikipedia.org/wiki/GitLab>
5. <https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git>
6. https://es.wikipedia.org/wiki/Control_de_versiones#Uso_de_ramas
7. <https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git>