Práctica de GIT

INTRODUCCIÓN AL CONTROL DE VERSIONES

Uso de un repositorio para el desarrollo de código en grupo

Tabla de contenidos

1 Introducción

Un sistema de control de versiones es un lugar donde se almacenan las distintas revisiones de los distintos ficheros usados en el desarrollo de una aplicación.

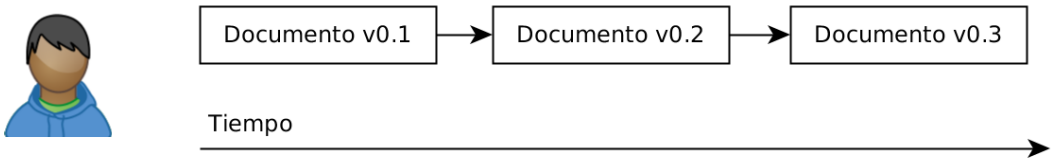


Ilustración 1. Método clásico de modificación local de documentos.

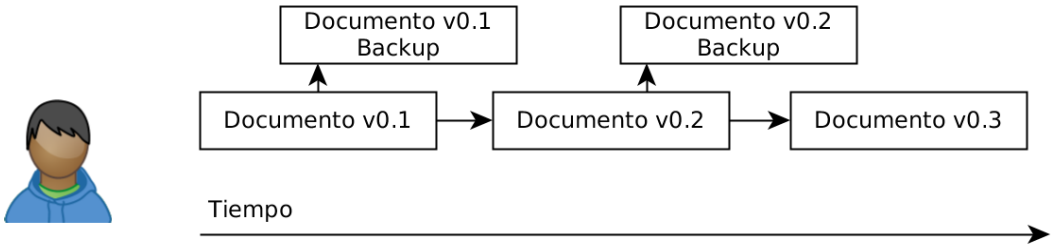


Ilustración 2. Método clásico de modificación local de documentos y control de versiones.

Si bien los métodos clásicos (trabajar en local y hacer backups de ficheros) parecen a priori totalmente funcionales y válidos, su utilidad en el desarrollo de aplicaciones es mínima y está totalmente desaconsejado.

El uso de un control de versiones permite guardar un historial de cambios sobre todos los ficheros relativos a un proyecto y la posibilidad de realizar comparaciones entre diversas versiones, así como restaurar uno o varios ficheros de anteriores versiones, lo que posibilitará cambiar la versión de desarrollo de un proyecto software de forma inmediata y segura.

Además, un sistema de control de versiones es especialmente útil cuando sobre un mismo proyecto trabajan varias personas.

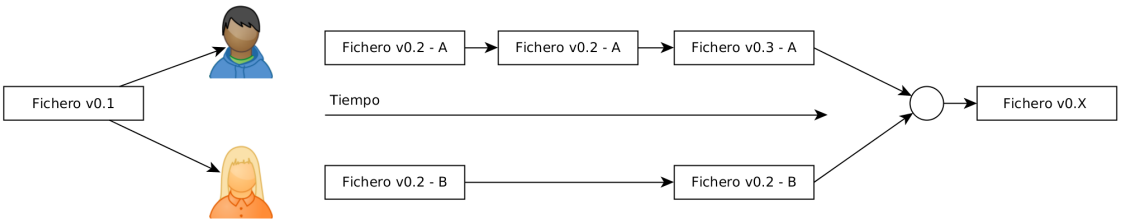


Ilustración 3. Método clásico de modificación de documentos de forma local por más de una persona y posterior mezcla manual de los cambios.

Vamos a comenzar definiendo un repositorio y los principales tipos de repositorio en los sistemas de control de versiones.

1.1 Repositorio

Es el lugar donde se almacena la información relativa al control de versiones. Contiene todos los datos de todas las versiones de los ficheros de los diferentes proyectos existentes.

Algunos sistemas de control de versiones usan como método de almacenamiento una base de datos, otros un conjunto de archivos, y algunos una combinación de los dos métodos.

1.1.1 Repositorio Centralizado

Son aquellos sistemas de control de versiones que disponen de un único servidor para mantener los archivos versionados. Todos los clientes de este repositorio descargarán los ficheros desde este lugar.

Al ser un parte fundamental del sistema de control de versiones, es recomendable que el repositorio se encuentre situado en una máquina segura y que se cree un sistema robusto que aplique una política de copias de seguridad adecuada a las características del grupo de trabajo.

Ejemplos de este tipo de repositorios (*CentralizedVersion Control Systems* o CVCSs en inglés): CVS o Subversion.

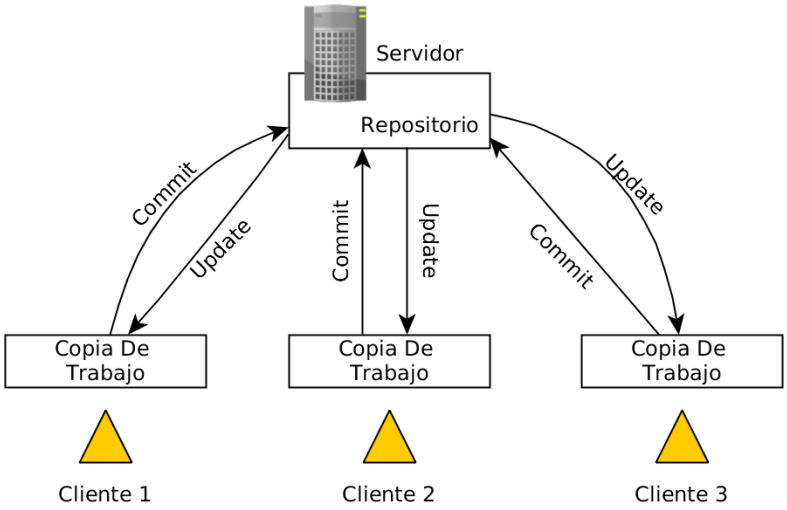


Ilustración 4. Sistema de Control de Versiones Centralizado.

1.1.2 Repositorio Distribuido

Son aquellos sistemas de control de versiones dondecada uno de los clientes dispone de su propia copia del repositorio con todas las versiones hasta el momento. Para formalizar el trabajo con este tipo de repositorios es necesario definir en algún momento un repositorio central que se considerará el "oficial" de ese proyecto.

Una de las formas más extendidas de oficializar la línea de trabajo de un proyecto es definirlo en repositorios en la nube como GitHub o Bitbucket.

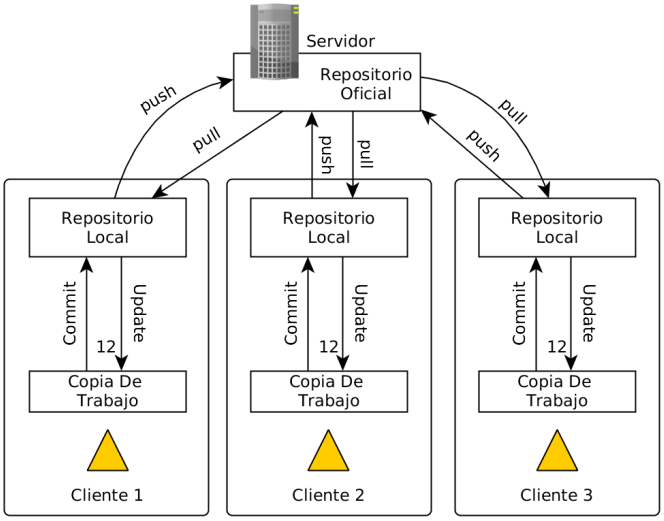
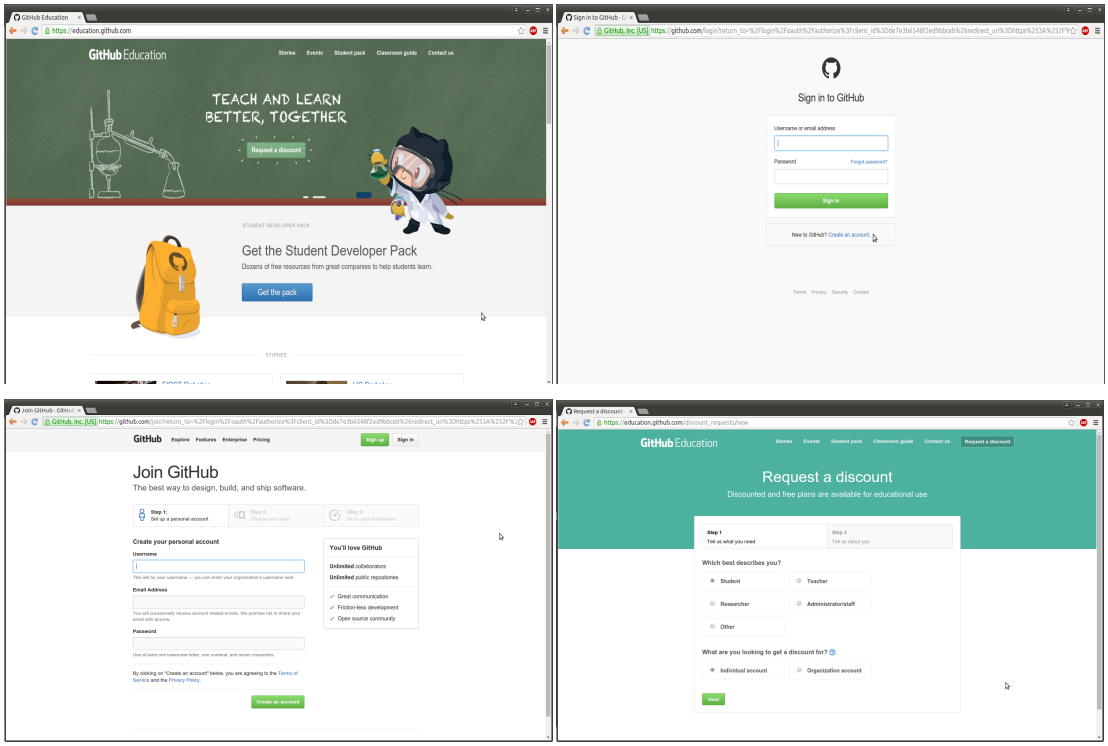


Ilustración 5.Sistema de Control de Versiones Distribuido

1.2 Creación de cuenta en GitHub

El primer paso en esta práctica consiste en darse de alta en GitHub con las credenciales de unileon (@estudiantes.unileon.es). Es obligatorio utilizar dichas credenciales ya que sin ellas no se dará de alta en el repositorio de la asignatura.



2 Terminología Básica en Git

2.1 Repositorio**local**

Una vez instalado Git, es posible crear un repositorio local de control de versiones en cualquier *path* de tu ordenador. El repositorio contendrá una carpeta llamada `.git` que contiene herramientas para la gestión e información acerca del repositorio.

```
user@Watermelon:~/tmp/Repositorios/Ejemplo1$ tree .git/
.git/
├── branches
├── COMMIT_EDITMSG
├── config<- Configuración local del repositorio
├── description<- Fichero que contiene la descripción del repositorio
├── HEAD<- Contiene puntero/hash a un commit
├── hooks<- Scripts con acciones a ejecutar ante eventos
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-rebase.sample
│   └── update.sample
├── index
├── info
│   └── exclude<- Ficheros a excluir del repositorio
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects<- Contiene información del repositorio
│   ├── 5f
│   │   └── 2d5920121a800b78b8f71b563cd1c8a260b5c9
│   ├── 61
│   │   └── eb5c252b77c5123a7035c6c42bc50781349924
│   ├── 6e
│   │   └── 01b5ba7dc9974f31036cc9ef356ce484c58941
│   ├── info
│   └── pack
├── refs<- Contiene puntero/hash a un commit
│   ├── heads
│   │   └── master
│   └── tags
```

2.2 Copia de trabajo local

Las copias de trabajo locales son una copia de todos elementos ubicados en el repositorio necesarios para trabajar con nuestro proyecto.

En el caso de proyectos de baja o media envergadura, la copia de trabajo local se compondrá del código fuente y de todos los elementos necesarios para el despliegue total del proyecto. Para grandes proyectos, la copia local contendrá un subconjunto de los elementos que integran el proyecto. En estos casos, las copias locales se denominan directorios de trabajo (*workingdirectories*) o, simplemente, espacios de trabajo (*workspaces*).

2.2.1 Relación entre repositorio local y copia de trabajo local

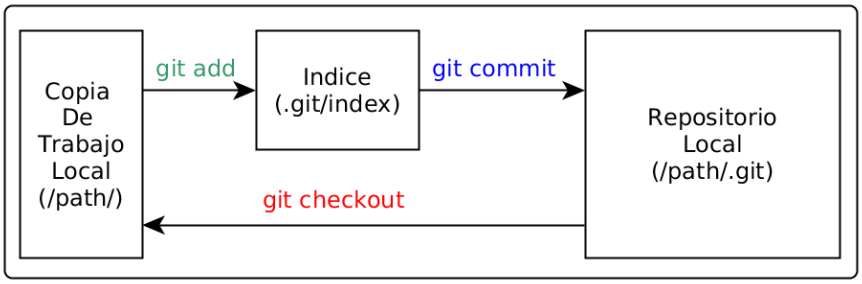


Ilustración 6. Pasos para introducir elementos de la copia de trabajo local al repositorio local.

2.2.2 Relación entre repositorio local y repositorio remoto

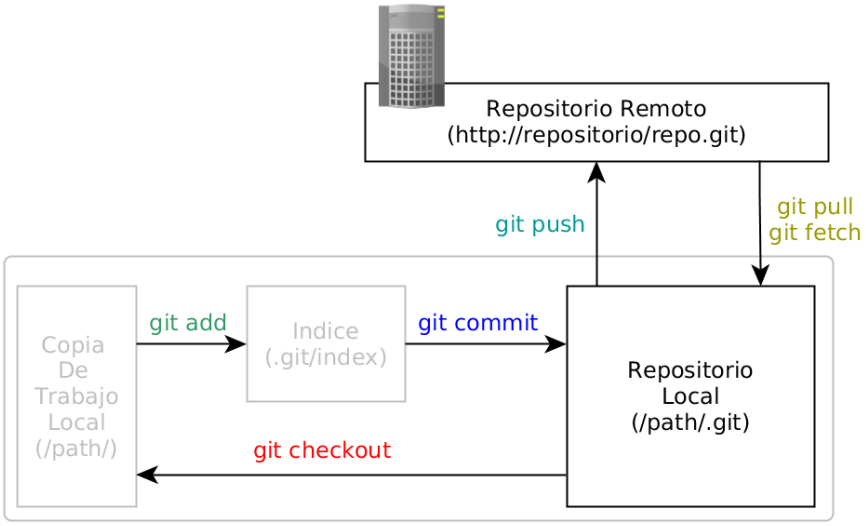


Ilustración 7. Pasos para enviar los cambios del repositorio local al repositorio remoto.

2.3 Committing

El conjunto de cambios realizados en la copia local como consecuencia del desarrollo de un proyecto debe reflejarse en el repositorio. Para ello los sistemas de control de versiones poseen la herramienta *commit* que actualiza en el repositorio los cambios realizados en el conjunto de ficheros que pertenecen al proyecto.

En Git es necesario informar al repositorio previamente de los ficheros que se han modificado o que se han añadido (*add*) y que necesitan ser contemplados por el control de versiones.

2.4 Checkout

Una vez fijados los cambios en el repositorio (*commits*), es posible descargar una versión previa de un fichero seleccionando entre las distintas versiones existentes.

Para ello será necesario revisar el listado de acciones realizadas desde la creación del repositorio.

2.5 Log

Cada vez que hacemos *commit*, establecemos una marca temporal en el repositorio. Para revisar estas marcas, disponemos de varias herramientas que permiten obtener nombre del autor, su correo, fecha y un hash asociado a dicho *commit*.

```
user@Watermelon:~/tmp/Repositorios/Ejemplo1$ git log
commit904794ce230158f077f97a7382f2a62901b1fdcf
Author: Trol <Trol@trol.es>
Date:   ThuJan 21 16:47:11 2016 +0100

    Un nuevo commit

commit c59bb1ceb18b13f7d5c7662bc53a87d67c5a977d
Author: Trol <Trol@trol.es>
Date:   ThuJan 21 16:46:14 2016 +0100

    Un monton de texto

commit 6e01e5ba7dc9974f31076bc9ef356ce484c58941
Author: Trol <Trol@trol.es>
Date:   ThuJan 21 16:44:52 2016 +0100

    New File onrepository
```

```
fran@Watermelon:~/tmp/Repositorios/Ejemplo1$  git show904794ce230158f077f97a7382f2a62901b1fdcf
commit 904794ce230158f077f97a7382f2a62901b1fdcf
Author: Trol <Trol@trol.es>
Date:   ThuJan 21 16:47:11 2016 +0100

    Un nuevo commit

diff --git a/README b/README
index cce0074..ace5738 100644
--- a/README
+++ b/README
@@ -8,3 +8,12 @@ onlyone file, butthisprocessappliesequallytoanynumber of files and/orfo
 in yourproject.

+
+Don’t be concernedbyallthescarymessagesGitthrows at you at thispoint. The
+worldisnotabouttoend!
+
```

2.6 Update

Cuando hay varias personas trabajando en un mismo conjunto de ficheros, pueden darse condiciones para que aparezcan distintos cambios en ellos. Esta situación se produce cuando varios usuarios realizan *commits* en sus repositorios locales y posteriormente suben estos cambios al repositorio remoto oficial. También puede producirse cuando repositorios locales son tomados como repositorios de origen de datos.

Con la herramienta *update,* los sistemas de control de versiones permiten actualizar en la copia local la última versión de los ficheros presentes en el repositorio.

Evidentemente existe un problema potencial en este mecanismo. ¿Qué ocurre cuando dos personas realizan cambios en el mismo fichero al mismo tiempo? La solución aplicada depende del tipo de sistema de control de versiones.

Git ofrece diferentes herramientas para obtener los ficheros de un repositorio remoto <*gitfetch*> y <*gitpull*>. La principal diferencia consiste en que el primero descarga aquellos ficheros remotos que han sido modificados o añadidos en el repositorio remoto y el segundo recupera e intenta unir en tu repositorio local los cambios realizados.

3 Comenzando con Git

3.1 Creación de un repositorio

Para la creación de un repositorio se necesita un directorio vacío para almacenar los datos. Posteriormente se deberá ejecutar un conjunto de comandos específicos para configurar los parámetros de Git correctamente.

Supongamos que estamos usando los siguientes directorios `/home/user/git-repositorio` (para Unix).

Inicialización

```
$ mkdir -p /home/user/git-repositorio1
$ cd /home/user/git-repositorio1
$ git init
```

Configuración

```
$ git config --global user.name "Pepe Garcia"
$ git config --global user.email pepe@example.com
```

Otras configuraciones:

```
$ git config --global core.editor <emacs,nano, vim,...>
```

Cuestionario 1.

a) Identifica los directorios y ficheros que se crean dentro de la carpeta del repositorio. ¿Qué utilidad tiene cada uno de ellos? ¿Se pueden configurar?¿Cómo?

b) ¿Para qué sirve el fichero `gitconfig`? ¿Cual es la ruta?

c) ¿Hay alguna forma de visualizar los valores de los parámetros de configuración anteriores?

3.2 Creación de un proyecto de trabajo local

Para la ejemplificación de la creación de un proyecto, vamos a suponer que tenemos dos ficheros y que deseamos importarlos al repositorio previamente creado.

Primero creamos un directorio local donde tengamos ubicados los ficheros que componen nuestro proyecto y que nombraremos como `Practica_Git`. Dentro del directorio creamos los dos ficheros que necesitamos para nuestro ejemplo:

- Fichero `Dia.txt`, cuyo contenido es el siguiente:

  Lunes

  Martes

  Miércoles

  Jueves

  Viernes

- Fichero `Numero.txt` siendo su contenido:

  Cero

  Uno

  Dos

  Tres

  Cuatro

A continuación vamos a añadir los ficheros al repositorio. Para esto necesitaremos lanzar desde nuestro terminal

```
$ git add Practica_Git/
```

Cuestionario 2.

1. Lanza el comando de visualización del estado
2. Observa el resultado de dicho comando.

El siguiente paso consiste en añadir los ficheros al repositorio. En este caso necesitaremos lanzar el comando

```
$ git commit -m "Hemos introducido los dos ficheros base a nuestro repositorio local"
```

La opción `-m` permite asociar un mensaje a la importación. Es recomendable incluir siempre un mensaje aclaratorio para indicar qué clase de importación se está realizando.

Si no se incluye el parámetro -m, se desplegará el editor de archivos asociado a git para introducir el mensaje.

Está **totalmente desaconsejado** realizar *commits* **sin un mensaje conciso de lo que se está grabando en el repositorio.**

3.3 Creación de un proyecto de trabajo remoto

El siguiente paso consiste en trasladar nuestro repositorio local al repositorio remoto.

Para ello vamos a utilizar el servicio de GitHub. GitHub es una plataforma que nos permite alojar el servidor remoto de forma gratuita. El inconveniente de este servicio es que los repositorios son **públicos**. Los beneficios son muchos, ya que dispone de

herramientas para desarrollo colaborativo como wiki, seguimiento de errores/*issues* del software, herramienta de revisión de código y un visor de ramas de desarrollo.

El primer paso en este caso es crear una cuenta en GitHub con tus credenciales de unileon.

El segundo paso es crear un repositorio de prueba para disponer de un origen "oficial" de datos. Este segundo paso puede realizarse de dos formas:

1. Utilizando el interface de GitHub.
2. Desde línea de comandos aprovechando el API v3 de GitHub podemos crear un repositorio remoto en nuestro repo.

```
$ curl -u 'USER_GITHUB' https://api.github.com/user/repos -d
'{"name":"garlic-p1", "description":"Creando un repositorio de prueba
con mensaje"}'
```

Una vez realizado esto hay que trasladar nuestro repositorio local al repositorio remoto. La ruta del repositorio en GitHub debe incluir el usuario remoto, por ejemplo: `https://user_ule@github.com/user_ule/Repo.git`

```
1. $ git remote add origin <Ruta HTTP/HTTPS del repositorio en GitHub>
2. $ git push -u origin master
```

3.4 Comenzando a trabajar con un proyecto

Ahora vamos a desplegar nuestro repositorio "oficial" de esta práctica en un nuevo directorio. Para realizar el despliegue del contenido del repositorio en local se debe usar:

```
$ mkdir -p /home/user/git-repositorio2
$ git clone "Ruta HTTP/HTTPS del repositorio en GitHub"
```

3.5 Realizando cambios

Supongamos que realizamos una modificación en el fichero `Dia.txt`, añadiendo dos líneas al final del mismo:

Lunes

Martes

Miércoles

Jueves

Viernes

Sábado

Domingo

Cuestionario 3.

Con el comando de visualización del estado obtén el estado de los ficheros en el repositorio local.

Observa los elementos del mensaje mostrado por el comando.

3.6 Actualizando repositorios

Los pasos para actualizar el repositorio son 3, dos asociados al repositorio local y uno asociado al repositorio remoto

Repositorio Local

```
1. git  add  Dia.txt
2. git  commit -m "Mensaje realizado"
```

Repositorio Remoto

```
1. git  push
```

Después de usar el comando *$git commit* se puede usar el *$git log* para comprobar que los cambios han sido realizados en el repositorio. Después de realizar el *$git push* también puedes revisar el log en GitHub.

3.7 Ejercicios propuestos

Ejercicio 1.

a) Crea un repositorio en tu cuenta de GitHub llamado PROG1_ule_nombreusuarioule. Dicho fichero contendrá al menos el fichero README.md

b) Ahora clona tu repositorio en local y añade un fichero llamado PROG1.txt. El fichero debe contener 10 líneas y cada línea contiene una palabra cualquiera de la RAE (real academia de la lengua) y un enlace asociado.

P.e.:

1. lámpara - http://dle.rae.es/?id=MqxCKuY

2. cañería - http://dle.rae.es/?id=7G1uVB0

3…

c) Realiza un *commit* de dicho fichero y súbelo a tu repositorio GitHub.

d) Comprueba que el fichero está en el repositorio remoto de GitHub y no solo en el repositorio local.

Ejercicio 2.

d) Clona el repositorio de un compañero.

e) Añade tus palabras a su fichero PROG1.md

f) Realiza un *commit*.

g) Realiza un *push* a su repositorio.

Nota 1: Todo el mundo debe tener en su repositorio un *commit* de un compañero.

Nota 2: Todo el mundo debe realizar al menos un *commit* sobre el repositorio de uno o más compañeros.

4 Bibliografía y enlaces de interés

[1] GitHub, https://github.com/

[2] Manual de GitHub, https://git-scm.com/book/es/v1/Empezando