

ENTORNOS DE DESARROLLO

PRÁCTICA 1



Bitbucket



GitLab

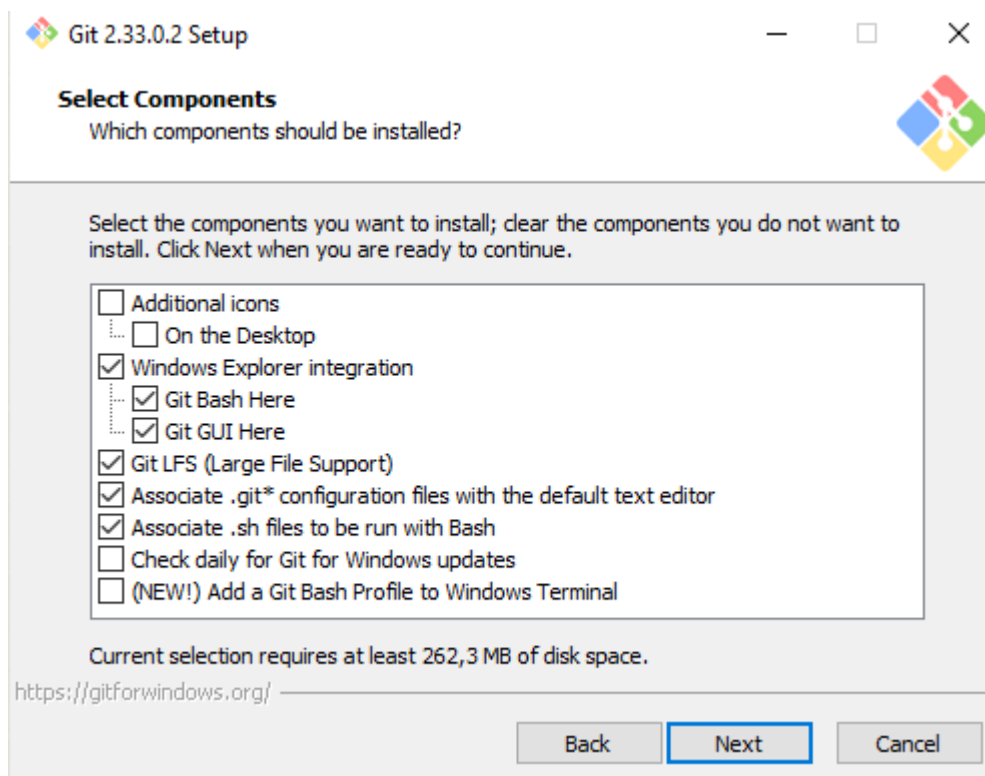
Colegio Calasanz 2022-2023

Instalación

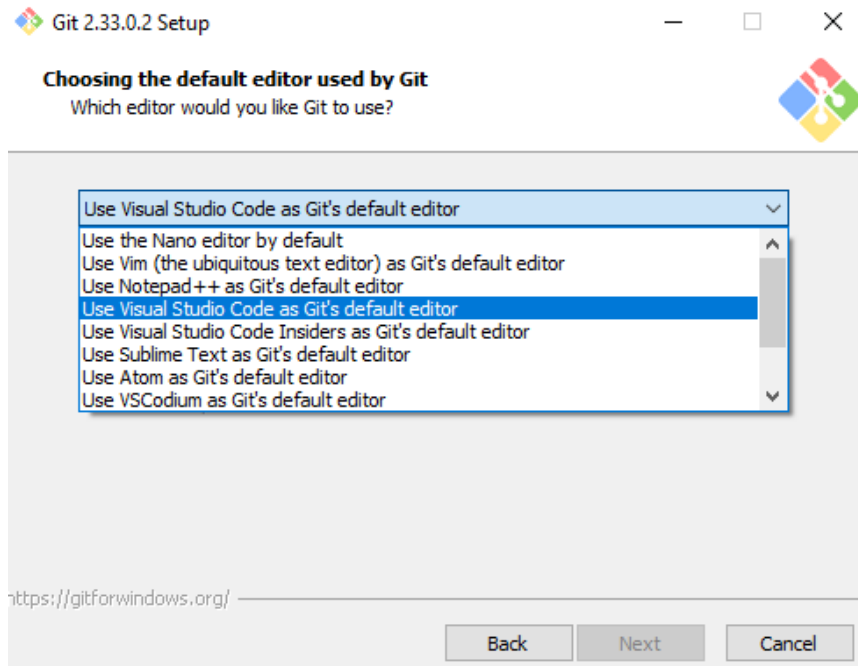
Para hacer uso de un repositorio Git, el primer paso es instalarlo. Se va a trabajar con Git en modo consola, por lo que todas las operaciones se harán a través de la terminal cmd, powershell de Windows o Git Bash. Los pasos para instalar y realizar una configuración básica de git son:

Descargar git, la versión que se corresponda con el sistema operativo utilizado, de la página <https://git-scm.com/downloads>.

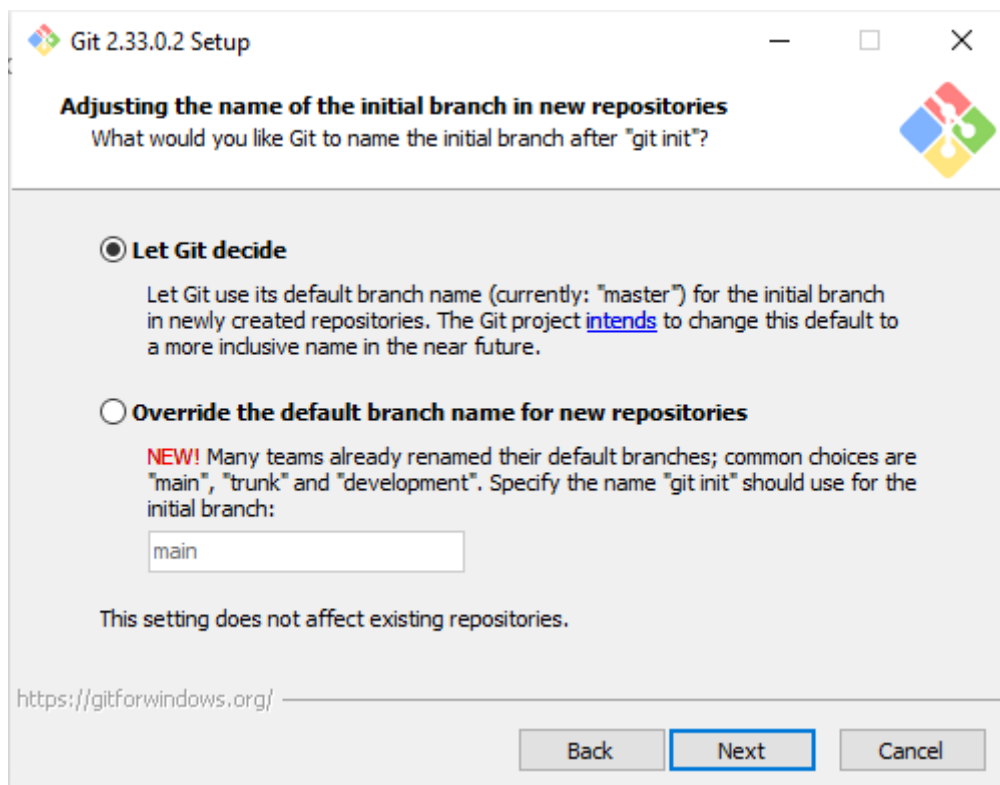
Lanzar el ejecutable y dejar las opciones seleccionadas por defecto. Las opciones de Windows Explorer integration deberían estar marcadas (se marcarían si no lo estuvieran).



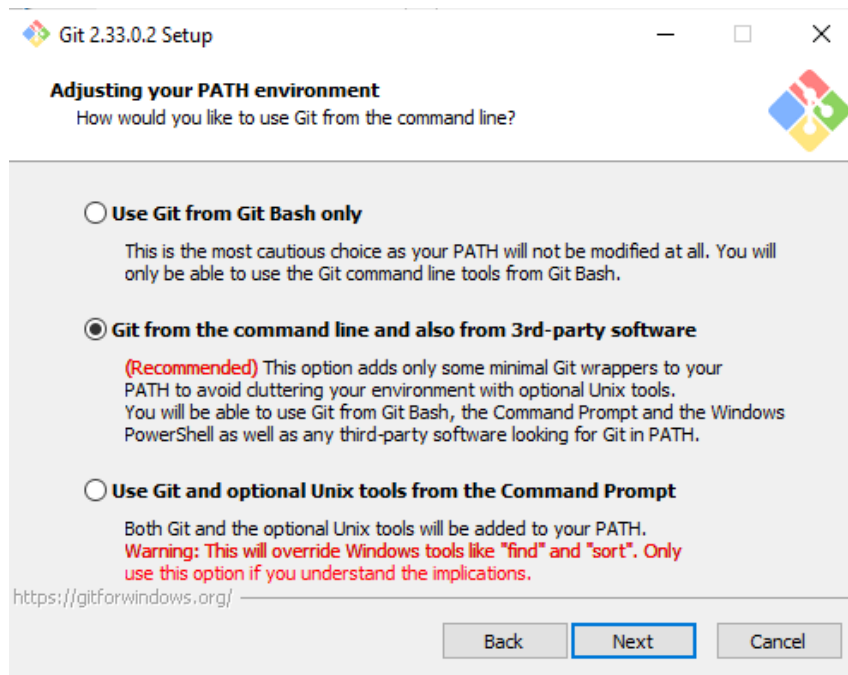
Nos dará la opción de seleccionar algún editor por defecto si así lo deseamos.



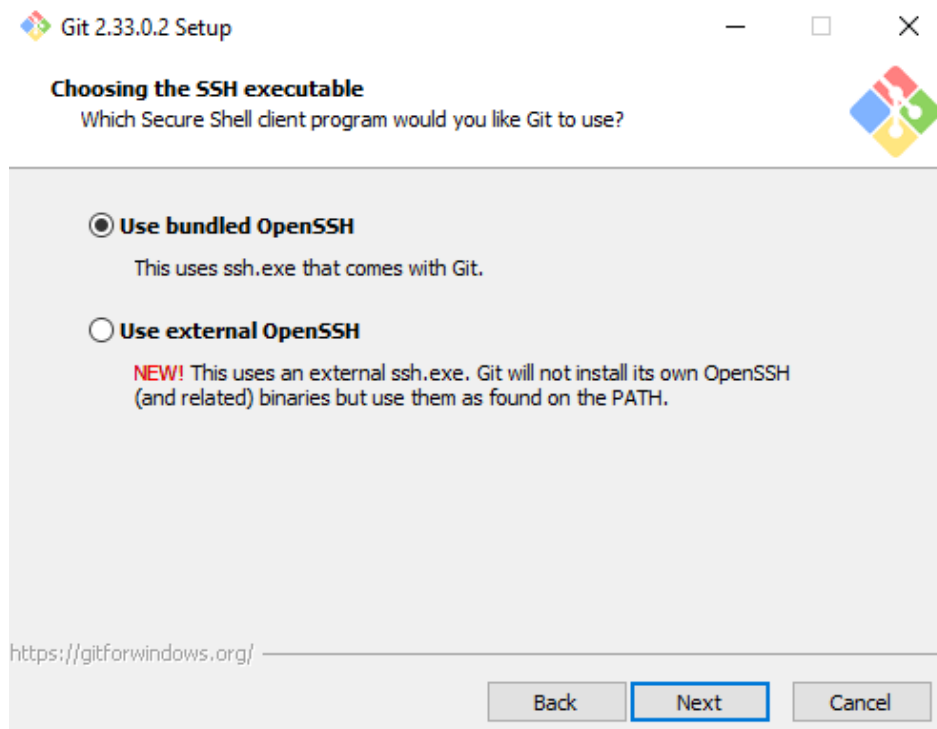
Podemos cambiar el nombre de la rama principal de nuestro proyecto. En caso de no hacerlo git por defecto la denominará “master”.



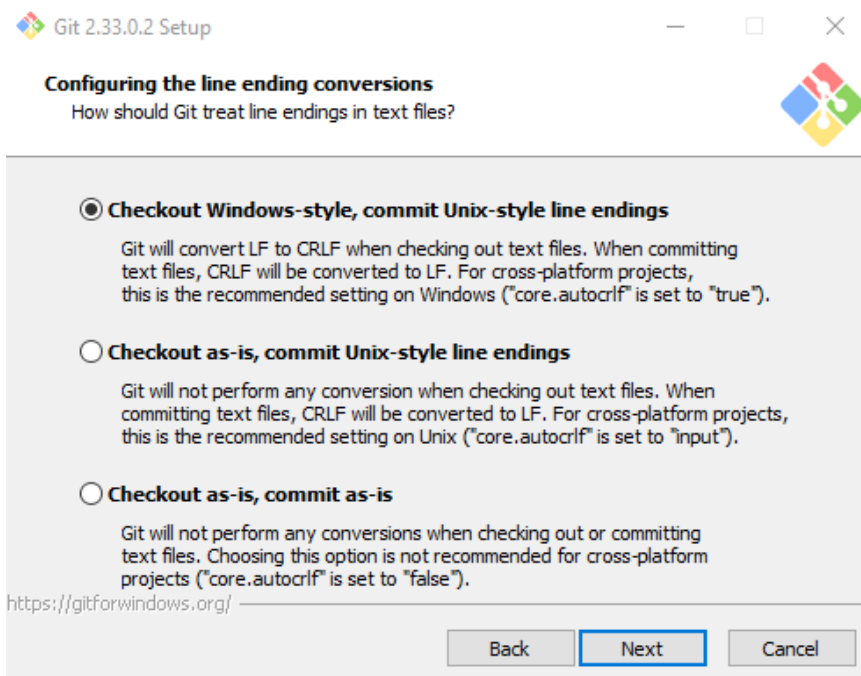
Dejamos la opción recomendada en la elección de la línea de comandos para así tener una mayor flexibilidad, sin modificar ciertos comandos.



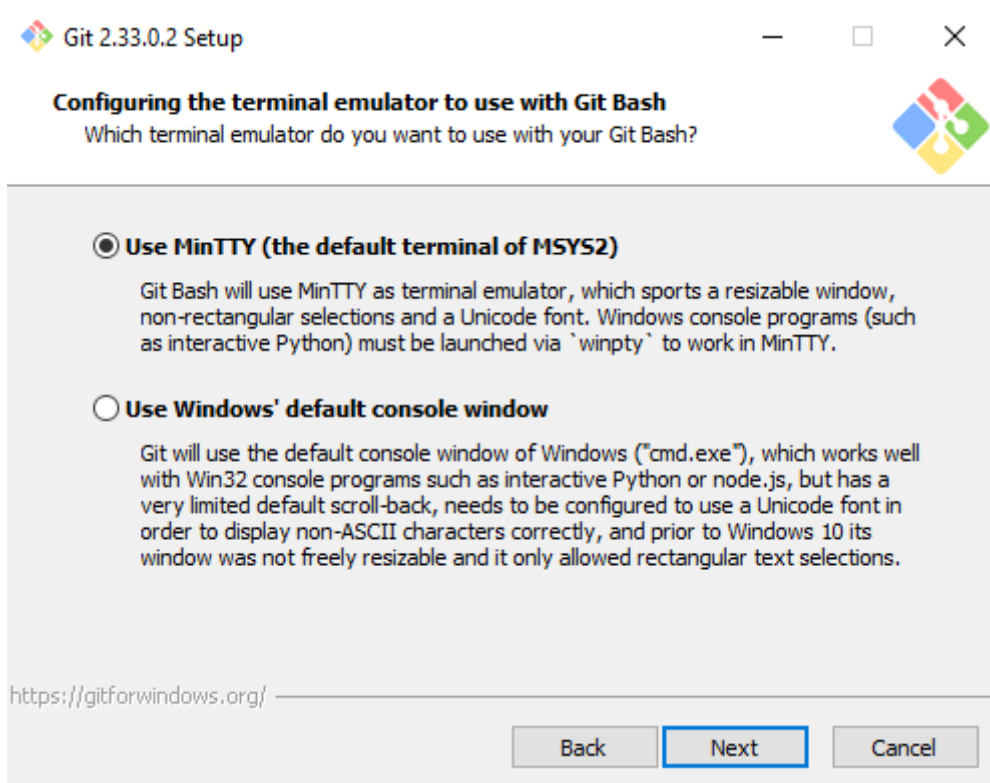
Seleccionamos la biblioteca de conexión segura de Git, para que el propio Git se encargue de todo.



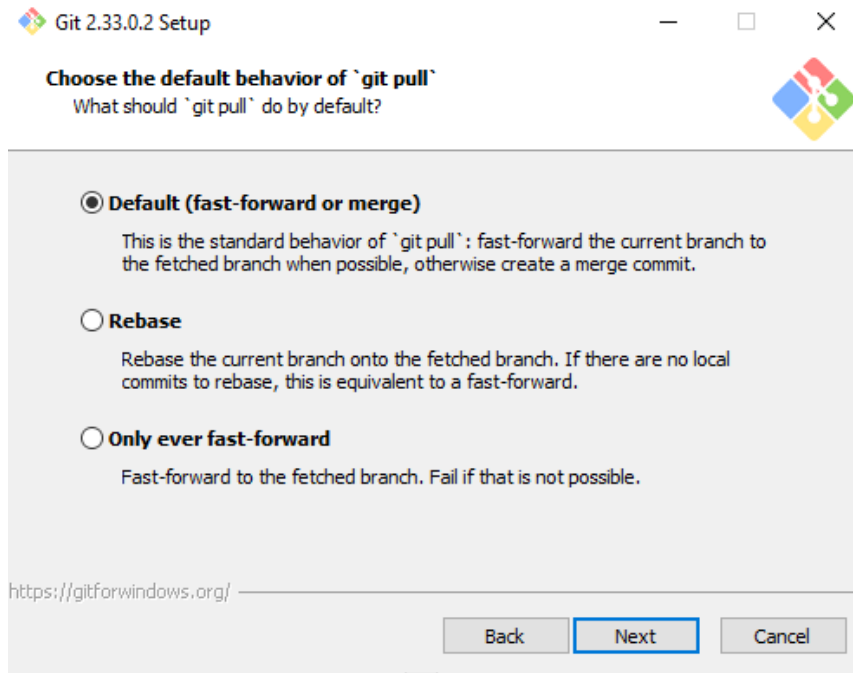
Puede que otras personas de nuestro proyecto tengan otros sistemas operativos distintos al nuestro, por tanto marcamos la opción con la que Git hace la conversión desde el sistema operativo al estilo Linux.



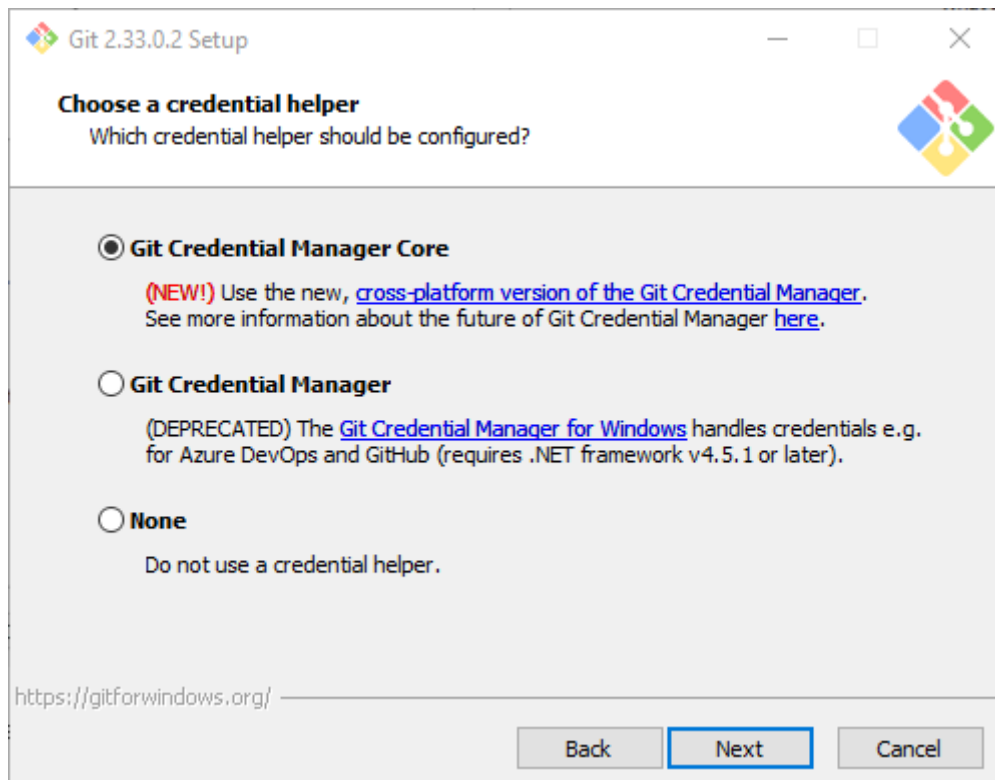
Seleccionamos la terminal que Git nos ofrece, puesto que será más intuitiva que la línea de comandos, ya que guardará una mayor relación con la terminal de Linux.



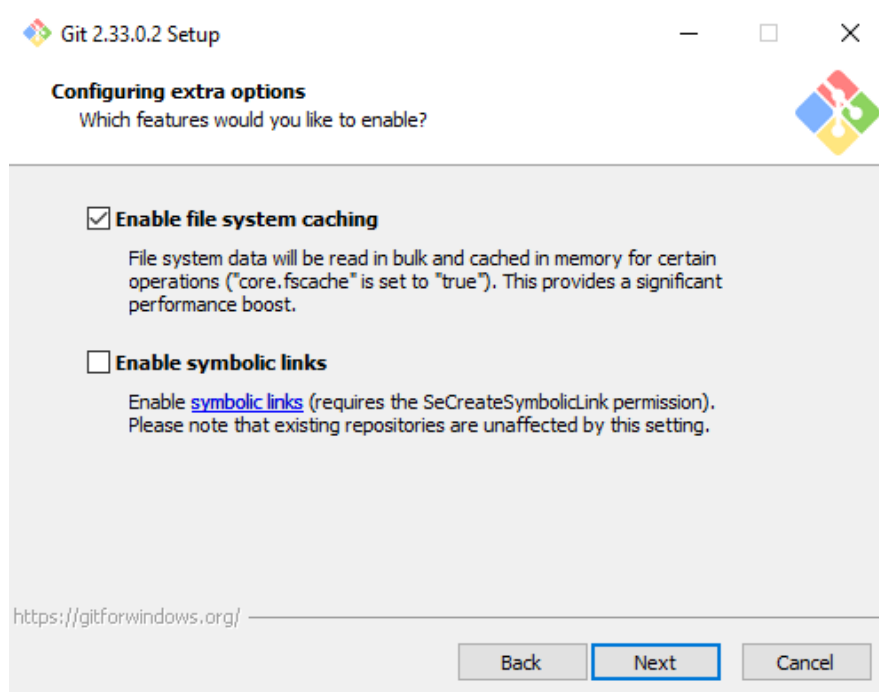
Mantenemos la opción del “pull”, para que realice un “fetch” y un “merge”, puesto que es la manera más segura.



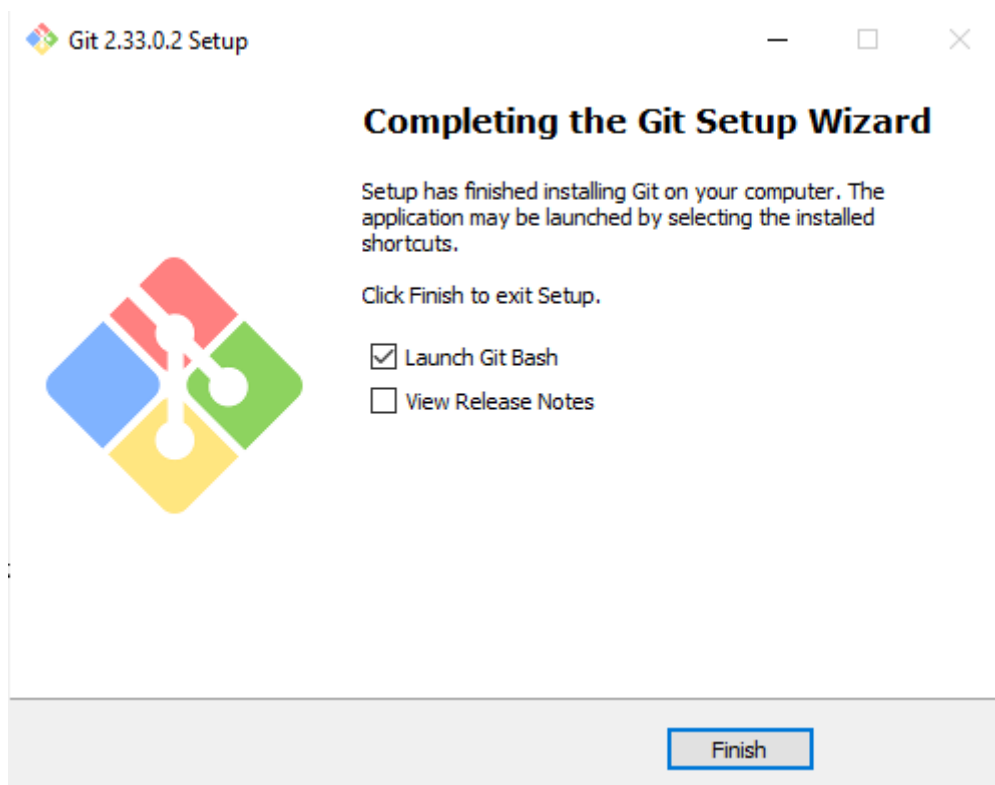
Mantenemos la opción que Git nos ofrece acerca del gestor de credenciales.



Seleccionamos la siguiente configuración extra y obviamos las configuraciones experimentales.



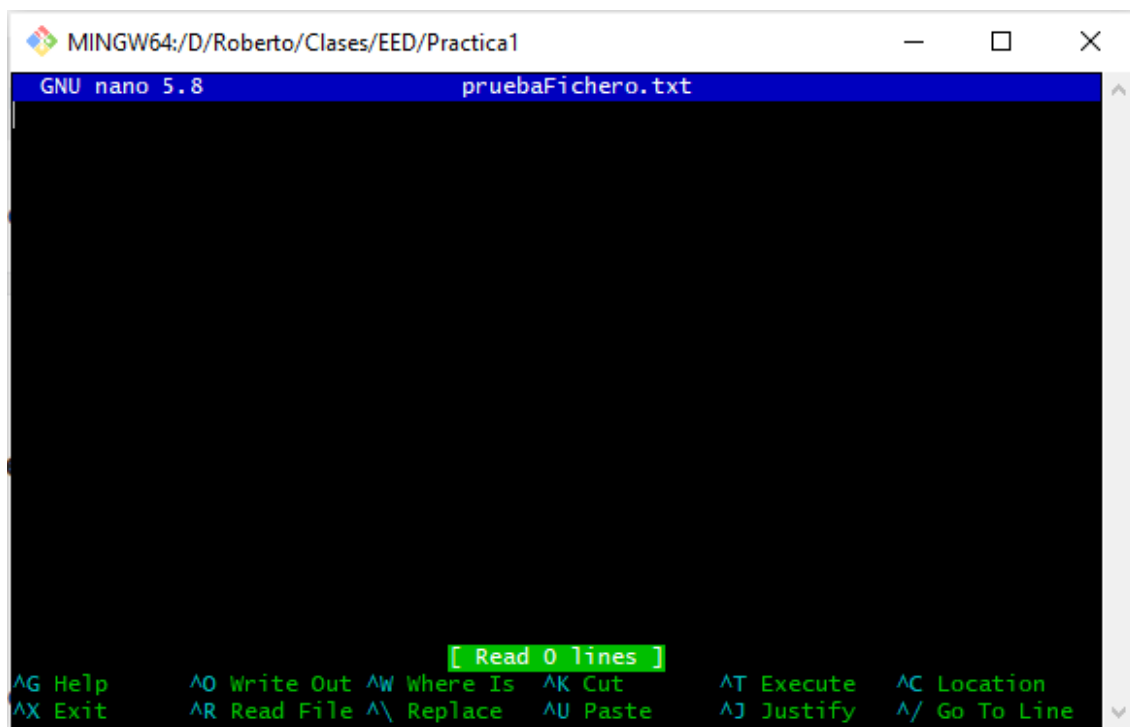
Una vez instalado, seleccionamos la opción para que se ejecute desde la “Git Bash” y acabamos la instalación.



Comandos Linux

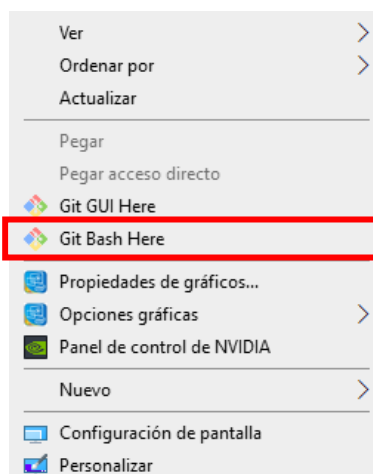
Como ya se ha comentado, Git ofrece integración tanto en un terminal de Windows como en uno de Linux (Bash). Los siguientes comandos están centrados en el manejo de Bash:

- **Pwd:** muestra la ruta del archivo actual.
- **Cd:** Nos permite la navegación por los distintos directorios de nuestro ordenador. Podemos trabajar con rutas absolutas o mediante sus atajos.
 - **Cd .. :** (dos puntos), volvemos un directorio hacia atrás (hacia la ruta raíz)
 - **Cd:** Para volver a nuestro directorio raíz
 - **Cd - :** (guión), para acudir al anterior directorio en el que nos encontramos.
 - **Cd /(ruta absoluta):** vamos directamente al directorio que hemos indicado en la ruta. Por ej. "`cd /D/Roberto/Clases/EED/Practica1`".
- **Ls:** Muestra el contenido del directorio de trabajo en el que nos encontramos. `ls`
 - **ls -R:** también listará todos los archivos en los subdirectorios
 - **ls -a:** mostrará los archivos ocultos
 - **ls -al:** listará los archivos y directorios con información detallada como los permisos, el tamaño, el propietario, etc.
- **Clear:** limpia la terminal de comandos.
- **Cp:** Sirve para copiar archivos del directorio actual a un directorio diferente. Su forma de uso es: `cp (nombre del archivo) (ruta donde lo queremos pegar)`. Por ej. "`cp pruebaCp.JPG /D/Roberto/Clases/EED`"
- **Mv:** Comando utilizado para mover archivos. También puede ser utilizado para cambiar el nombre de un archivo. Los argumentos son similares al comando `cp`. `Mv (nombre del archivo) (ruta donde lo queremos mover)`. Por ej. "`mv pruebaCp.JPG /D/Roberto/Clases/EED`". En caso de querer renombrar un archivo: `mv (antiguo nombre) (nuevo nombre)`. Por ej. "`mv pruebaCp.JPG pruebaNew.JPG`"
- **Mkdir:** Nos crear un nuevo directorio. "`mkdir (Directorio)`". Por ej. "`mkdir carpeta`".
- **Rmdir:** Eliminar directorios vacíos, solo vacíos. Por ej. "`rmdir carpeta`".
- **Rm:** Utilizado para la eliminación de archivos y directorios. "`rm [nombre_archivo]`" nos servirá para eliminar el archivo indicado. Por ej. "`rm pruebaNew.JPG`"
 - **Rm - r:** ¡CUIDADO! Elimina un directorio y todo el contenido de su interior. No permite volver atrás.
- **Touch:** Nos permite crear un archivo en blanco. "`touch [Nombre_archivo.extension]`". Por ej. "`touch pruebaFichero.txt`"
- **Nano:** Edición de archivos. "`nano [nombre_archivo]`". Por ej. "`nano pruebaFichero.txt`". Dentro de la ventana de uso de este editor podremos ver las distintas opciones que nos da. Por ej. "`ctr + X, para salir`".



Actividad 1. Uso de comandos Linux.

Abre la terminal de Git previamente instalada (Git Bash Here).



Realiza un documento, en el que se muestre la puesta en uso de algunos de los comandos vistos con anterioridad. Algunas de las tareas que puedes realizar son las siguientes:

- Comprueba la ruta en la que te encuentras.
- Muévete por los distintos directorios de tu ordenador, comprobando que se encuentra en su interior.
- Crear directorios
- Crear archivos

- Mover o copiar archivos a otro directorio.
- Editar archivos
- Eliminar archivos o directorios.

Recuerda que algunos comandos pueden ser “peligrosos”, por lo que úsalos con precaución.

Consejo: El uso del tabulador activa la función de autocompletar, lo cual nos puede ahorrar tiempo a la hora de escribir nombres de archivos.

Añade aquellas explicaciones, capturas u observaciones que consideres oportunas.

Git en local

Escribiendo git en la línea de comandos, mostrará algunas de las opciones que ofrece:

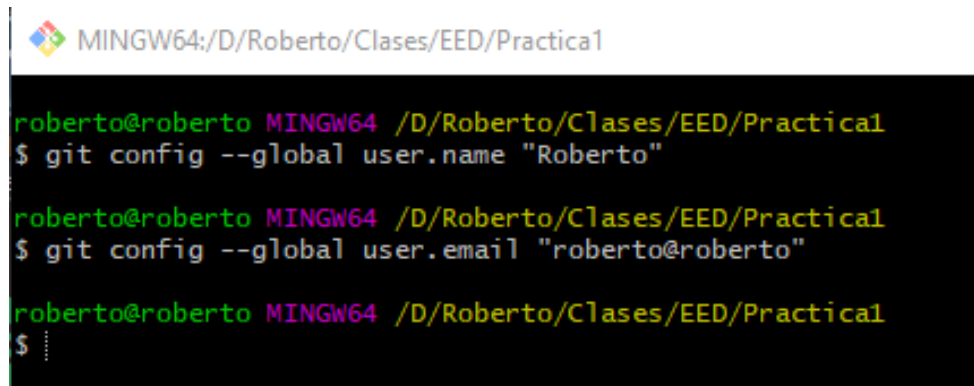
```
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      [--super-prefix=<path>] [--config-env=<name>=<envvar>]
      <command> [<args>]

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
  sparse-checkout    Initialize and modify the sparse-checkout

grow, mark and tweak your common history
  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
```

Ahora vamos a establecer una configuración inicial. Realmente lo único estrictamente necesario es indicar usuario y correo electrónico que se van a utilizar para realizar todas las operaciones con Git:

A terminal window with a black background and green text. The title bar shows a Windows logo and the path 'MINGW64:/D/Roberto/Clases/EED/Practica1'. The prompt is 'roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1'. Three commands are entered: 'git config --global user.name "Roberto"', 'git config --global user.email "roberto@roberto"', and a third line with a colon and a dollar sign.

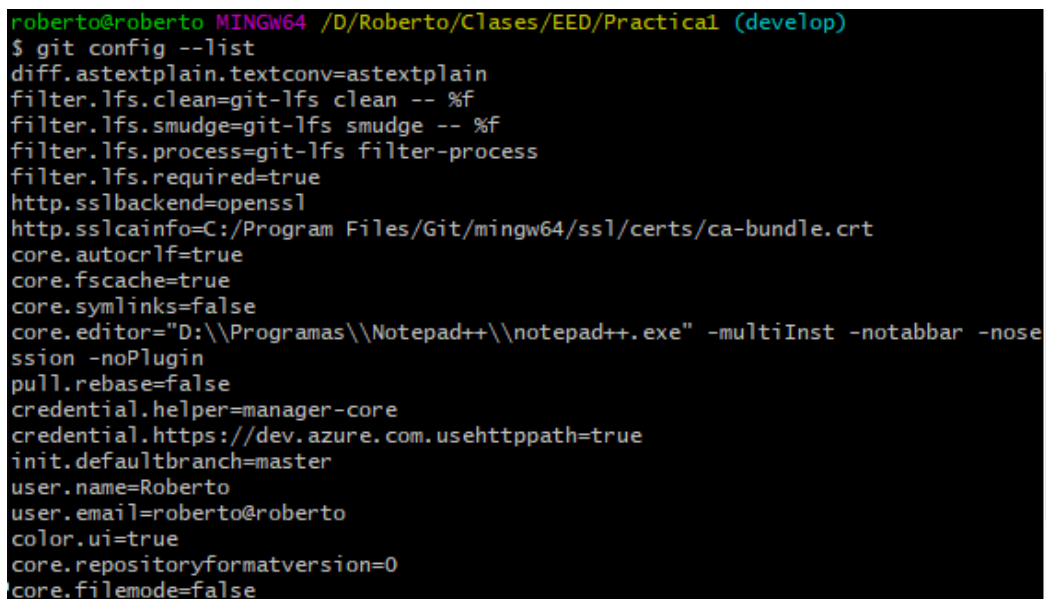
```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1
$ git config --global user.name "Roberto"

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1
$ git config --global user.email "roberto@roberto"

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1
$ :
```

Al indicar la opción *--global* estos datos serán para todos los repositorios con los que se trabaje. Si se quiere cambiar el usuario para un repositorio concreto, sería el mismo comando sin la opción *--global* y ejecutado dentro del directorio del repositorio en sí.

Para ver los datos de configuración podemos utilizar el comando *"git config --list"*

A terminal window with a black background and green text. The title bar shows a Windows logo and the path 'MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)'. The prompt is 'roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)'. The command 'git config --list' is entered, and its output is displayed.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor="D:\\Programas\\Notepad++\\notepad++.exe" -multiInst -notabbar -nosession -noPlugin
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Roberto
user.email=roberto@roberto
color.ui=true
core.repositoryformatversion=0
core.filemode=false
```

Partiendo de aquí hay otras opciones de configuración que, si bien no son imprescindibles para trabajar con git de forma general, una vez que se vaya conociendo Git pueden ser interesantes. Por ejemplo, hacer que se coloreen las diferencias en los commits para entender mejor los cambios con:

"git config -- global color.ui true"

Git ya incluye esta opción por defecto, no haciendo necesario hacer uso de este comando.

Una vez que ya hemos visto la configuración, se van a detallar una serie de comandos que permiten trabajar con Git, con un repositorio local, a través de la línea de comandos.

Git init

Comando para crear un repositorio local. Se ejecuta solo una vez, y con esa acción se indica que en el directorio en el que se esté situado en ese momento se llevará un control de versiones (esa carpeta se convierte en repositorio). Internamente el comando, al ejecutarse, crea un directorio oculto (.git) con la información que Git necesita para registrar las versiones.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1
$ git init
Initialized empty Git repository in D:/Roberto/Clases/EED/Practica1/.git/
```

Si se le pasa como parámetro un nombre, el repositorio local será una carpeta que se creará con dicho nombre en el directorio donde se encuentre. Por ej. *"git init PracticaGit"*. En este caso nos crearía la carpeta "PracticaGit" dentro de la carpeta que nos encontramos y este sería nuestro repositorio.

Para eliminar un repositorio utilizaremos el comando *"rm -rf .git"*, desde el repositorio (carpeta) donde hemos inicializado git.

Git status

Al ejecutar este comando se puede ver el estado actual del repositorio. Indicará los cambios que ha habido en el directorio (repositorio) desde la última versión realizada. Muestra los nuevos ficheros añadidos al repositorio o fuera de seguimiento, los modificados y los eliminados, desde el último commit.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
```

Git add

Con este comando se pueden incluir archivos o directorios al índice (pasarán al estado staged). Permite añadir archivos que están fuera de seguimiento (untracked) o que han sido modificados (modified) y prepararlos para el siguiente commit. *"git add [nombre_archivo]"*. Por ej. *"git add pruebaFichero.txt"*

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git add pruebaFichero.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git status
On branch master

No commits yet

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

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

Si no se indica nombre de fichero a subir ni de directorio, se añadirá todo lo modificado.

Con el comando “git add .” o “git add -A” se incluirán todos los elementos de la carpeta en la que nos encontramos.

Git commit

Comando para confirmar las modificaciones realizadas. Se creará una nueva versión de los documentos que estuvieran en estado staged. Es recomendable la inclusión de un mensaje descriptivo de la confirmación que se va a realizar.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git commit -m "commit inicial"
[master (root-commit) f1c20e8] commit inicial
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 pruebaFichero.txt
```

De todas las opciones que tiene commit, las dos más utilizadas son -m y -a:

- **git commit -m “comentario”**: indica que se va a añadir un mensaje
- **git commit -a**: Indica que se van a realizar las acciones de add y commit, en un solo paso, de todos los cambios. Al usar la opción -a hay una excepción, y es que cuando se trata de un fichero nuevo (no existente en el repositorio) no se hará commit de él, en ese caso si habrá que hacer git add primero.

Cabe la posibilidad de combinar estos comandos. Por ej. “git commit -am “comentario”, nos añadirá el fichero y el comentario.

Actividad 2. Primeros pasos con Git

Añade esta actividad al fichero de la actividad 1.

En este ejercicio se van a probar los comandos vistos. Todos los puntos del ejercicio se harán a través de la línea de comandos:

- Crea un nuevo repositorio llamado Biblioteca y, una vez creado, muestra su contenido.
- Comprueba el estado del repositorio.
- Crea un fichero libros.txt con el siguiente contenido:
 - Libros prestados: 54
 - Libros disponibles: 129
 - Libros reservados: 21
- Comprueba de nuevo el estado del repositorio.
- Añade el fichero al índice, volviendo a comprobar el estado del repositorio.
- Realiza commit de los últimos cambios con el mensaje "Inventario libros 05/10/2021" y mira de nuevo el estado del repositorio.
- Cambia el fichero libros.txt para que contenga la siguiente información:
 - Libros prestados: 120
 - Libros disponibles: 210
 - Libros reservados: 3
- Mostrar los cambios con respecto a la última versión guardada en el repositorio.
- Hacer un commit de los cambios con el mensaje "Inventario actualizado a 07/10/2021".

Git branch

Este comando nos servirá para crear una nueva rama de trabajo. *"git branch [nombre_rama]"*. Por ej. *"git branch develop"*.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (master)
$ git branch develop
```

Este comando utilizado sin opciones nos servirá para ver las ramas que tenemos.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (master)
$ git branch
develop
* master
prueba
```

Con la opción *-d*, *"git branch -d [nombre_rama]"* podremos eliminar una rama. Por ej. *"git branch -d prueba"*.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (master)
$ git branch -d prueba
Deleted branch prueba (was deac194).
```

Con el comando “*git show-branch*”, podremos todas las ramas y los commits que se han ido realizando sobre ellas.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git show-branch
* [develop] cambio desa
! [master] segundo commit
--
* [develop] cambio desa
*+ [master] segundo commit
```

Git checkout

Si se tienen más de una rama en un repositorio, para poder cambiar de una a otra se utilizará este comando seguido del nombre de la rama. Esto automáticamente actualizará el directorio de trabajo con el contenido de la rama a la que se está cambiando.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git branch
develop
* master

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git checkout develop
Switched to branch 'develop'

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git branch
* develop
master
```

Git diff

Nos permitirá ver las diferencias entre el índice en el que nos encontramos y el repositorio de trabajo que le indiquemos. “*git diff [árbol_trabajo]*”. Por ej. “*git diff master*”.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git diff master
diff --git a/pruebaFichero.txt b/pruebaFichero.txt
index e69de29..5c1b149 100644
--- a/pruebaFichero.txt
+++ b/pruebaFichero.txt
@@ -0,0 +1 @@
+holá
```

Podemos ver como se muestra el fichero en el cual tenemos la diferencia, y cuál es esa línea que hemos añadido.

Git merge

Haciendo uso de este comando vamos a poder fusionar dos ramas incluyendo los cambios de unas en la otra. Su forma de uso es “*git merge [rama_a_fusionar]*”. Por ej. “*git merge develop*”

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git merge develop
Updating deac194..bacfff4
Fast-forward
 pruebaFichero.txt | 1 +
 1 file changed, 1 insertion(+)
```

Nota: Nos debemos colocar, mediante checkout, en la rama en la que queremos añadir los cambios. Es decir, en este caso queremos añadir los cambios a la rama master, por tanto nos tenemos que posicionar en ella y a continuación hacer el merge.

Con la opción “`git merge [rama_a_fusionar] -m “comentario”`”, podemos añadir un comentario a esta fusión.

Git tag

Con el comando “`git tag [nombre etiqueta]`” asignaremos un nombre al momento actual en el que esté apuntando la cabeza (HEAD) de la rama en la que nos encontramos. Por ej. “`git tag v1.0`”, desde la línea master, asignará esta etiqueta al último commit de la línea master, puesto que no hemos apuntado a ningún otro commit en concreto. Conviene realizar este tipo de acción cuando tenemos algún lanzamiento (release) de nuestro producto, para indicar esa versión funcional de nuestro producto.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git tag v1.0
```

Git log

Cuando se realiza un commit, se está creando en el repositorio una nueva versión del fichero en cuestión. Con “`git log`” es posible ver el histórico de operaciones que se han realizado. También es posible ver las etiquetas que hemos asignado.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git log
commit bacfff44885ad9ab24bcdafb31fdf9e30e099fbd (HEAD -> master, tag: v1.0, develop)
Author: Roberto <roberto@roberto>
Date: Mon Oct 4 12:58:48 2021 +0200

    cambio desa

commit deac194fe7e2b06125c568ac14f50bda254c4171
Author: Roberto <roberto@roberto>
Date: Mon Oct 4 12:44:19 2021 +0200

    segundo commit

commit f1c20e8779be4b233f34ae5d621c0cd7a159e673
Author: Roberto <roberto@roberto>
Date: Mon Oct 4 12:21:58 2021 +0200

    commit inicial
```


Si ejecutamos el comando `"git log --oneline"`, se mostrará el histórico resumido.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git log --oneline
bacfff4 (HEAD -> master, tag: v1.0, develop) cambio desa
deac194 segundo commit
f1c20e8 commit inicial
```

Actividad 3. Creando ramas

Añade al fichero con las dos actividades previas esta actividad.

Partiendo del repositorio Biblioteca de la actividad 2:

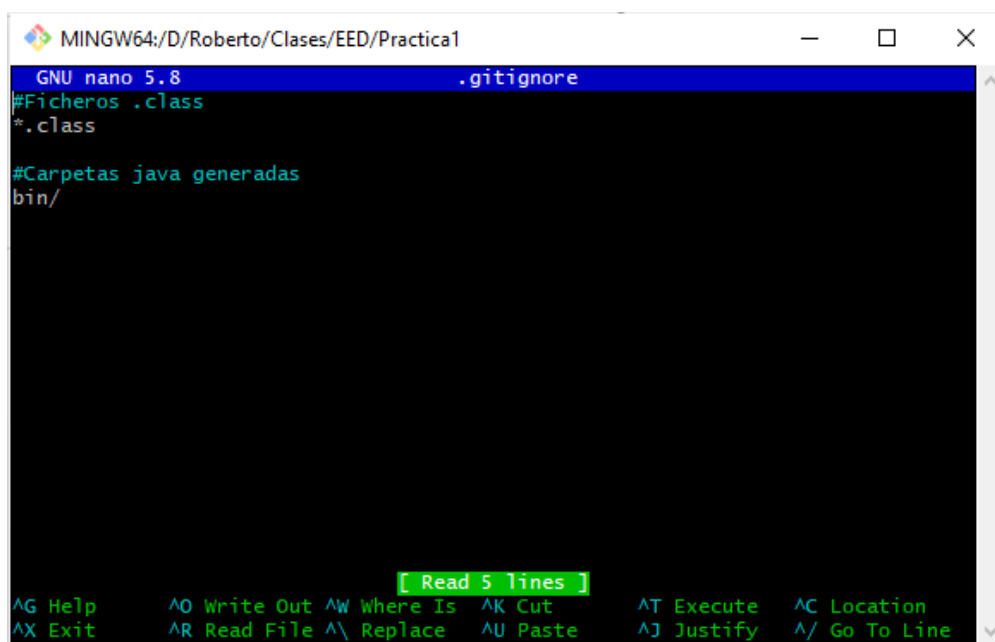
- Crea una rama de desarrollo a partir de la rama master.
- En esta rama de desarrollo Crea un segundo fichero llamados `revistas.txt`. Su contenido será:
 - Revistas prestadas: 27
 - Revistas disponibles: 31
 - Revistas reservadas: 6
- Súbelo al repositorio de desarrollo con el comentario "Desarrollo inventario revistas 08/10/2020"
- Mira el historial y el historial resumido
- Revisa las diferencias entre la rama de desarrollo y la master
- Haz una fusión de la línea de desarrollo y la línea master con el comentario "Incurción inventario revistas 08/10/2020 en producción"
- Pon una etiqueta en la línea master que indique que es la versión 1.0 de nuestro proyecto.
- Visualiza el estado del repositorio y el log del mismo.

Ignorar archivos

Cuando se está trabajando en proyectos de cierta entidad, es bastante común que éstos contengan archivos o directorios de los que no se quiere hacer un seguimiento. Por ejemplo, los ficheros `.class` de java.

Git ofrece la posibilidad de crear el fichero `.gitignore`, que contendrá los ficheros que se quieren excluir del control de versiones y sincronización con el repositorio remoto.

En el fichero los datos se indicarán de la siguiente manera:



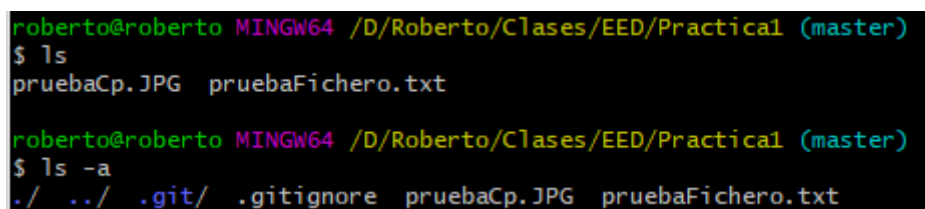
The screenshot shows a terminal window titled 'MINGW64:/D/Roberto/Clases/EED/Practica1'. Inside, the GNU nano 5.8 editor is open, editing a file named '.gitignore'. The content of the file is as follows:

```
.gitignore
#Ficheros .class
*.class

#Carpetas java generadas
bin/
```

At the bottom of the window, there is a status bar with various keyboard shortcuts: AG Help, AO Write Out, AW Where Is, AK Cut, AT Execute, AC Location, AX Exit, AR Read File, A\ Replace, AU Paste, AJ Justify, and A/ Go To Line. A green highlight is visible over the text '[Read 5 lines]'.

Para la visualización de este archivo en nuestro repositorio tendremos que mostrar los ficheros ocultos puesto que sino no será posible ver si está.



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

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ ls
pruebaCp.JPG  pruebaFichero.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ ls -a
./  ../  .git/  .gitignore  pruebaCp.JPG  pruebaFichero.txt
```

Git rm y restore

Comando para eliminar archivos de las versiones. Permite eliminar del directorio de trabajo archivos (ya confirmados) que no se quiere, por alguna razón, que sigan en el proyecto. Este proceso se realiza en dos pasos: primero hay que borrarlos de git (esto lo añadirá automáticamente al índice) y seguidamente confirmar para que la eliminación sea efectiva. Si se elimina un fichero del directorio de trabajo también desaparecerá del repositorio local.

Se utiliza el comando “*git rm [nombre_fichero]*”. Por ejemplo “*git rm borrar.txt*”.

```

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ ls
borrar.txt  pruebaCp.JPG  pruebaFichero.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git rm borrar.txt
rm 'borrar.txt'

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ ls
pruebaCp.JPG  pruebaFichero.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    borrar.txt

```

El comando “*git restore --staged [nombre_fichero]*” acompañado de “*git restore [nombre_fichero]*”, nos permitirá recuperar un archivo que hemos borrado y aún no hemos confirmado su borrado. Por ej. “*git restore --staged borrar.txt*” y “*git restore borrar.txt*”.

```

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git restore --staged borrar.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git status
On branch develop
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    borrar.txt

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

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ git restore borrar.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (develop)
$ ls
borrar.txt  pruebaCp.JPG  pruebaFichero.txt

```

En caso de borrar y hacer commit, podemos recuperar ese fichero desde algunos de los commit en los que se encontraba.

Para ello haremos checkout del fichero en su última versión “*git checkout 7c6dedc borrar.txt*”, en este caso 7c6dedc es el identificador que le da git a este commit (recuerdo: para ver los identificadores “*git log --oneline*”).

Después utilizaremos el comando “*git restore --staged borrar.txt*”, añadiremos de nuevo el fichero al repositorio y podremos volver a hacer commit.

Aquí abajo se muestra la secuencia completa.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git checkout 7c6dedc borrar.txt
Updated 1 path from 4940cb3

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ ls
borrar.txt  pruebaCp.JPG  pruebaFichero.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   borrar.txt
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git restore --staged borrar.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git status
On branch develop
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    borrar.txt
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git add borrar.txt

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   borrar.txt
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practica1 (develop)
$ git commit -m "fichero recuperado"
[develop 5fdf197] fichero recuperado
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 borrar.txt
```

Actividad 4. Borrando archivos

Añade al fichero con las tres actividades previas esta actividad.

Partiendo del repositorio Biblioteca:

- Vuelve a la rama de desarrollo e incluye un fichero con el nombre películas.txt
- Añade y guarda este fichero en esta rama.

- Parece que el desarrollo de las películas no tiene sentido, por lo que nos piden eliminar este fichero, pero como sabemos que desde la dirección de la biblioteca las normas cambian muy a menudo, no hacemos commit de este cambio.
 - Efectivamente, ahora la dirección quiere continuar con este desarrollo y nos piden que añadamos la siguiente información.
 - Revistas prestadas: 0
 - Revistas disponibles: 10
 - Revistas reservadas: 0
 - Guardamos los cambios en el repositorio.
 - Días después nos dicen que definitivamente no se va a continuar este desarrollo puesto que el número de películas es muy escaso y no tendrá éxito. En este momento eliminamos el fichero y hacemos commit.
 - De repente un donante anónimo decide donar su colección de películas y la biblioteca comienza a promocionar esta iniciativa y la gente comienza a reservar. Nos dicen que añadamos la siguiente información:
 - Revistas prestadas: 0
 - Revistas disponibles: 172
 - Revistas reservadas: 25
- Es decir, tendremos que recuperar el fichero previo y añadir los cambios.
- Para que la gente pueda reservar, habrá que pasar este desarrollo a producción. Una vez ya hemos recuperado es el momento de fusionar este desarrollo con la rama master, y además añadir la etiqueta de versión 1.1

Git en remoto

En este tema ya se ha explicado cómo funciona un sistema de control de versiones distribuido. Hasta el momento se ha visto cómo trabajar en un repositorio local, pero aún no como enlazar el repositorio local de cada uno de los desarrolladores de un proyecto con el repositorio remoto del mismo.

Sincronizar un repositorio local con uno remoto, supone que se puedan añadir las modificaciones que hay en el primero a al segundo y, de la misma forma, que se puedan llevar

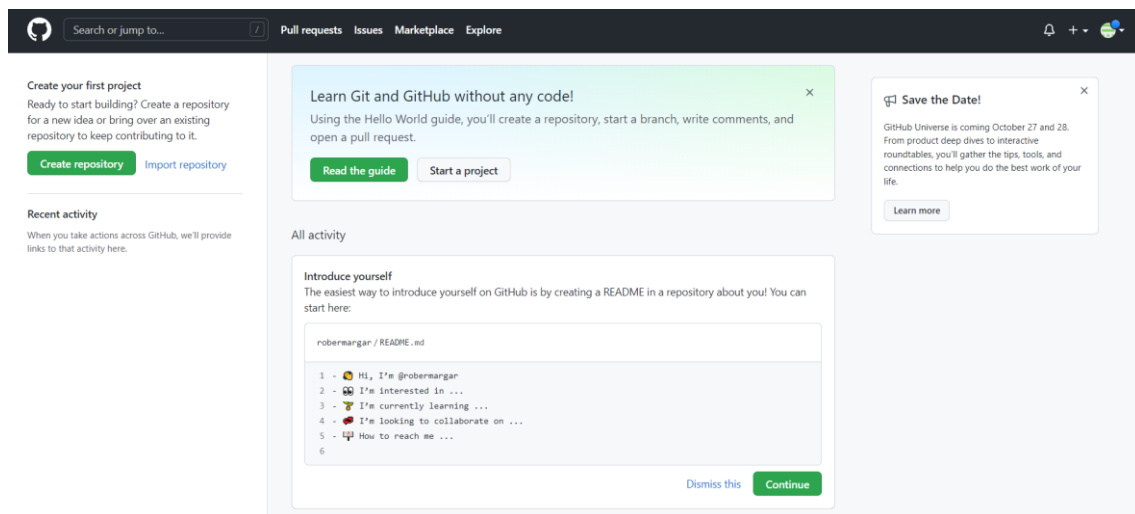
los últimos cambios subidos al remoto, por cualquier otro desarrollador, al repositorio local propio.

GitHub

Para la realización de las actividades en remoto vamos a utilizar GitHub (<https://github.com/>).

En primer lugar, tendremos que registrarnos. Si alguno tiene cuenta, puede utilizar esa sin problema.

Una vez registrados veremos algo como esto.



Picaremos en “Start a Project” y crearemos un repositorio Privado. Se puede ver cómo nos da la opción de añadir un fichero .gitignore, entre otros, pero en esta ocasión no lo vamos a utilizar.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner * robermargar / Repository name * EntornosDesarrollo ✓

Great repository names are short and memorable. Need inspiration? How about [shiny-computing-machine](#)?

Description (optional)

☐ 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:
Skip this step if you're importing an existing repository.

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

☐ Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

[Create repository](#)

Git clone

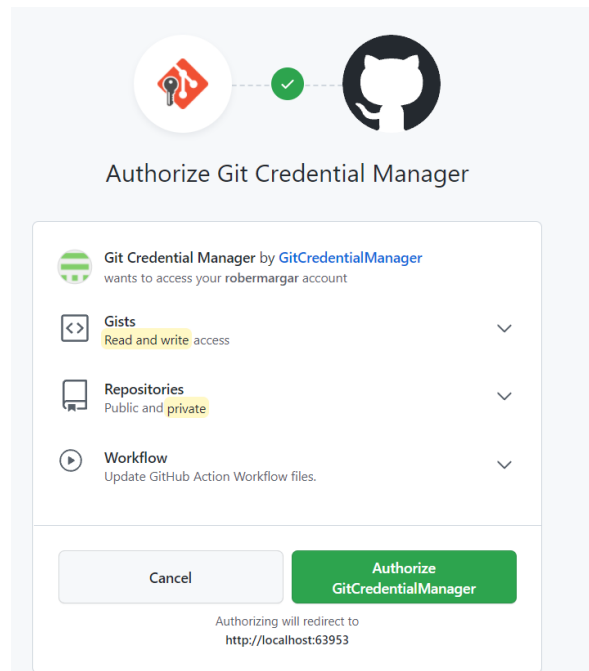
El primer paso para poder llevar a cabo esta sincronización, es obtener una copia del repositorio remoto.

Con el comando git clone, más la url del repositorio sobre el que se quiera trabajar, se puede crear una copia local del mismo. Es decir, este comando genera un repositorio local idéntico al remoto. Dicho repositorio se creará en una carpeta con el mismo nombre del repositorio remoto:

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED
$ git clone https://github.com/robermargar/EntornosDesarrollo.git
Cloning into 'EntornosDesarrollo'...

Excepción no controlada: System.ComponentModel.Win32Exception: El identificador
de la ventana no es válido
    en MS.Win32.ManagedWndProcTracker.HookUpDefWindowProc(IntPtr hwnd)
    en MS.Win32.ManagedWndProcTracker.OnAppDomainProcessExit()
    en MS.Internal.ShutdownListener.HandleShutdown(Object sender, EventArgs e)
warning: You appear to have cloned an empty repository.
```

En este proceso nos pedirá que validemos la clonación



Si quisiéramos haber clonado el repositorio con otro nombre de carpeta simplemente tendríamos que añadir el nombre deseado al final. Es decir, “*git clones [ruta] [nombre_carpeta]*”.

Una vez que el repositorio local esté creado, es decir se haya clonado el remoto, esa url utilizada se almacena con el alias **origin**, por lo que en acciones futuras, cuando se necesite hacer referencia a la url del repositorio remoto, solo habrá que poner su alias.

Git fetch

El comando `git fetch` descarga commits, archivos y referencias de un repositorio remoto a tu repositorio local. Esta acción la llevas a cabo cuando quieres ver en qué han estado trabajando los demás.

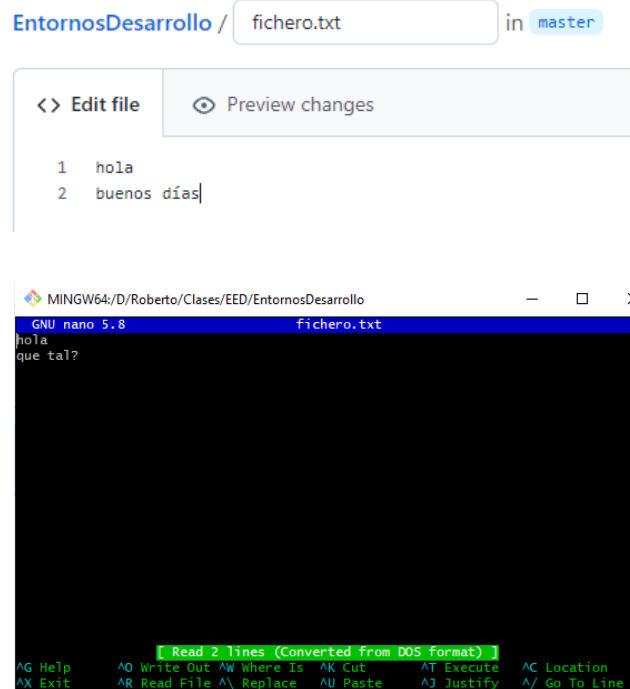
```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (develop)
$ git fetch origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 629 bytes | 4.00 KiB/s, done.
From https://github.com/robermargar/EntornosDesarrollo
    be1fa22..9a5b2b9  master    -> origin/master
```


Git pull

Para actualizar el repositorio local con los nuevos cambios que existan en el remoto, se ha de usar este comando. Es muy importante estar actualizado en el repositorio local para evitar conflictos a la hora de subir los cambios propios al remoto.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master)
$ git pull
Updating be1fa22..9a5b2b9
Fast-forward
 fichero.txt | 1 +
 1 file changed, 1 insertion(+)
```

Cuando se hace un pull y hay cambios en ese fichero en el repositorio local, se producen conflictos, por ejemplo:



Como los ficheros no son iguales se produce un conflicto que tendremos que resolver:

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master)
$ git pull
Auto-merging fichero.txt
CONFLICT (content): Merge conflict in fichero.txt
Automatic merge failed; fix conflicts and then commit the result.
roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master|MERGING)
$ nano fichero.txt
```

Dependiendo de la configuración que tenga git, se abrirá el fichero con los conflictos con el editor por defecto o habrá que editarlo con un editor cualquiera. En el documento con los

conflictos se mostrarán estos indicando que parte es la que está en el repositorio local y cuáles son los cambios del remoto:

```

MINGW64:/D/Roberto/Clases/EED/EntornosDesarrollo
GNU nano 5.8 fichero.txt
hola
<<<<<< HEAD
que tal?
=====
buenos días
>>>>>> ab87bce0a9eff5c207e8577fbe0cb2d2816e0c18

[ Read 6 lines (Converted from DOS format) ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line

```

```

roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master|MERGING
)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)
    both modified:   fichero.txt

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

```

Una vez hayamos resuelto el conflicto, como se puede ver en la imagen anterior, tendremos que realizar un commit del fichero que ya hemos guardado, para que todo vuelva a la normalidad.

```

roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master|MERGING
)
$ git commit -a
[master d284d15] Merge branch 'master' of https://github.com/robermargar/Entorno
sDesarrollo

roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master)
$ git pull
Already up to date.

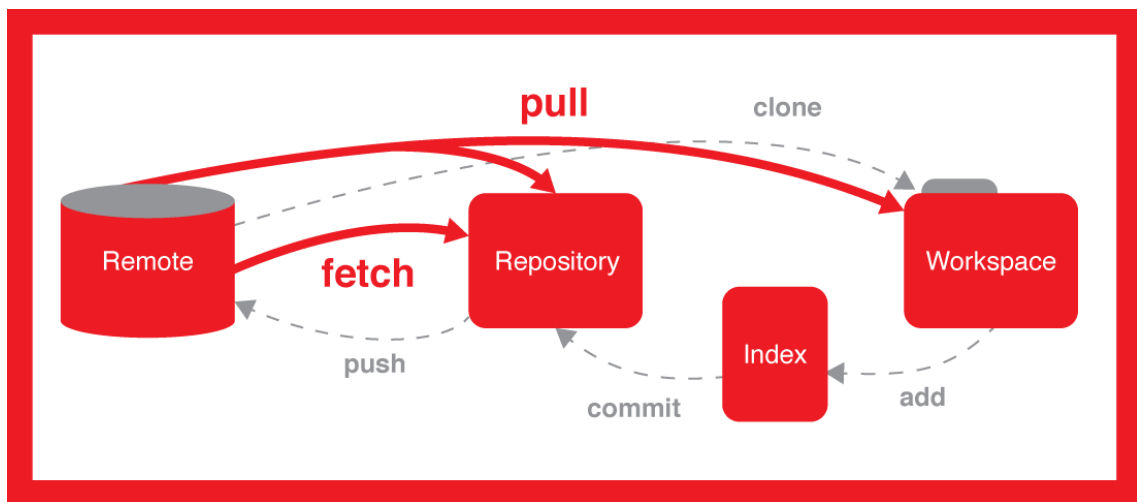
```

Git push

Utilizando este comando se puede subir el contenido del repositorio local al repositorio remoto. Hay que tener en cuenta que solo se sincroniza el contenido del repositorio local, no los cambios que existan en el directorio local. El comando es “*git push*”.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 205 bytes | 205.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/robermargar/EntornosDesarrollo.git
* [new branch]      master -> master
```

Para aclarar los conceptos, en esta imagen se puede ver cómo funcionan estos tres comandos



Git remote

El comando “*git remote*” nos permite ver los repositorios remotos vinculados al repositorio local. Si, además, se añade la opción -v, mostrará su dirección:

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master)
$ git remote
origin

roberto@roberto MINGW64 /D/Roberto/Clases/EED/EntornosDesarrollo (master)
$ git remote -v
origin https://github.com/robermargar/EntornosDesarrollo.git (fetch)
origin https://github.com/robermargar/EntornosDesarrollo.git (push)
```

Con el comando “*git remote add [nombre_repositorio] [url]*”, podemos asociar un repositorio local a uno remoto. Por ej. “*git remote add Practica1 https://github.com/robermargar/pruebaAdd.git*”. Hasta que no realicemos un “push”, “*git push Practica1*”, no veremos reflejados los cambios en nuestro repositorio remoto.

```
roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git remote add Practical https://github.com/robermargar/pruebaAdd.git

roberto@roberto MINGW64 /D/Roberto/Clases/EED/Practical (master)
$ git push Practical
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 37.71 KiB | 7.54 MiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/robermargar/pruebaAdd.git
* [new branch] master -> master
```

Con “*git remote rm [nombre_repositorio]*”, rompemos el enlace hacia el repositorio remoto. Esto no significa que hayamos eliminado el repositorio remoto, sino que simplemente ya no estará asociado a nuestro repositorio local.

Actividad 5. Trabajando en remoto

Crea un nuevo documento para añadir el trabajo referente a esta tarea.

Partiendo del repositorio Biblioteca (o el nombre que le hayas dado):

- Crea un nuevo proyecto en GitHub (con el nombre que desees, puede ser el mismo que tu repositorio en local).
- Añade tu repositorio local, previamente creado, con el que hemos estado trabajando, a este repositorio remoto que acabamos de crear.
- Modifica un fichero, y guarda, desde la página de GitHub.
- Comprueba el si hay cambios desde el repositorio local.
- Descarga esos cambios.
- Ahora modifica un fichero en tu área de trabajo y sube los cambios al repositorio local.
- Sube los cambios al repositorio remoto.