



GIT

Manejo básico de repositorios locales

Guillermo Palazón Cano




Sistemas de Control de Versiones


- ¿Por qué es ideal para el trabajo en equipo?
 - Permite trabajar diferentes programadores a la vez sobre un mismo proyecto (mismos archivos) trabajando con ramas.
 - Cada programador crea diferentes instantáneas de un archivo.
 - GIT fusiona el trabajo de diferentes programadores en sus diferentes ramas.
 - Pueden aparecer conflictos (colisiones)
 - GIT es capaz de resolver muchos de estos conflictos, aunque es posible que en ocasiones tengamos que tomar decisiones al respecto.





Git. Introducción

- Git es un software de control de versiones distribuido
- Desarrollado por Linus Torvalds → Software Libre.
- ¿Para qué sirve?
 - Facilita fases desarrollo y mantenimiento del proceso de desarrollo Software
 - Es útil para el trabajo individual pero especialmente para el trabajo en equipo.
- ¿Cómo facilita este desarrollo y mantenimiento?
 - Guarda registro de los cambios
 - Otra información que almacena: Persona que realiza el cambio, fecha, etc.

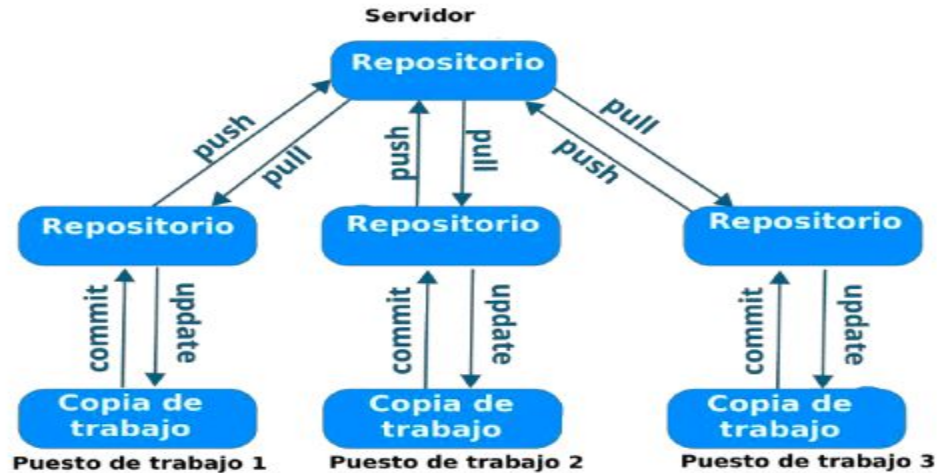
- 
- Aunque es posible utilizar GIT desde un IDE directamente de manera gráfica, es fundamental conocer y utilizar los comandos de su consola para poder sacar toda la utilidad del mismo ya que la mayoría de las interfaces gráficas solamente implementan una parte de las características de GIT.
 - Descarga de GIT
 - <https://git-scm.com/downloads>
 - En Linux en distribuciones basadas en Debian debería bastar con el comando: **apt-get install git**

- 
- Uno de los problemas que se encuentran las personas que necesitan colaborar con otros desarrolladores es tener un lugar (servidor) donde tener estos archivos.
 - Los sistemas de control de versiones centralizados como puede ser Subversion tienen un único servidor que contiene todos los archivos versionados y los clientes descargan los archivos desde ese lugar central.
 - Este ha sido el estándar para el control de versiones durante muchos años.
 - Esta configuración tiene muchas ventajas
 - Todos los desarrolladores saben hasta cierto punto en que están trabajando los demás desarrolladores del proyecto
 - El administrador tiene control detallado y es más fácil controlar todo el sistema de control de versiones

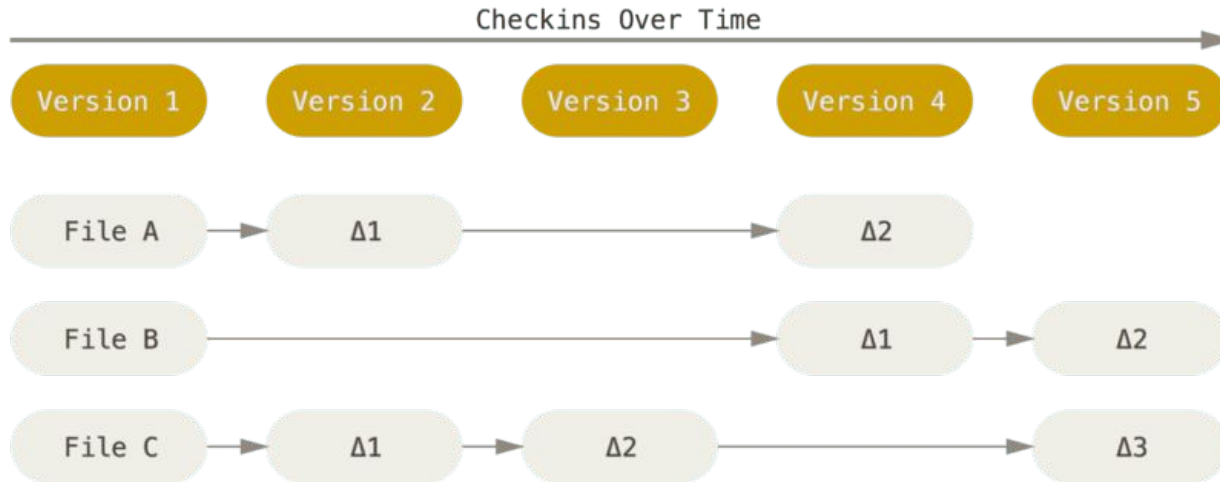
- 
- La principal diferencia entre GIT y cualquier otro sistema es la forma en la que manejan los datos.
 - La mayoría de sistemas (Subversión entre ellos) almacenan la información como una lista de cambios en los archivos. Almacenan los archivos y las modificaciones hechas a cada uno de ellos a través del tiempo.
 - GIT no almacena sus datos de esta forma, tiene un conjunto de copias instantáneas de un sistema de archivos miniatura.
 - Cada vez que confirmas un cambio o guardas el estado de un proyecto toma una foto del aspecto de todos tus archivos en ese momento y guarda una referencia a esa copia.
 - Para ser eficiente, si un archivo no se ha modificado GIT no lo almacena de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado.
 - GIT maneja por tanto los datos como una secuencia de copias instantáneas.

- 
- Sin embargo, una configuración centralizada también tiene serias desventajas
 - Punto único fallo que representa el servidor centralizado (si se cae durante un tiempo, nadie puede colaborar o guardar cambios en archivos que haya estado trabajando)
 - Si no se tienen copias de seguridad adecuadas se puede perder incluso la información de todo el proyecto (a excepción de lo que pueda tener los usuarios en sus máquinas locales)
 - Necesidad de disponer de Internet generalmente para que cada miembro pueda comunicarse con el servidor centralizado
 - Los sistemas de control de versiones **Distribuidos** ofrecen soluciones a los problemas que presentan los sistemas centralizados.
 - A diferencia de un sistema centralizado, los clientes no solamente se descargar una copia instantánea de los archivos, sino que se replica completamente el repositorio → Si el servidor deja de funcionar cualquier repositorio puede ser copiado en el servidor con el fin de restaurarlo.

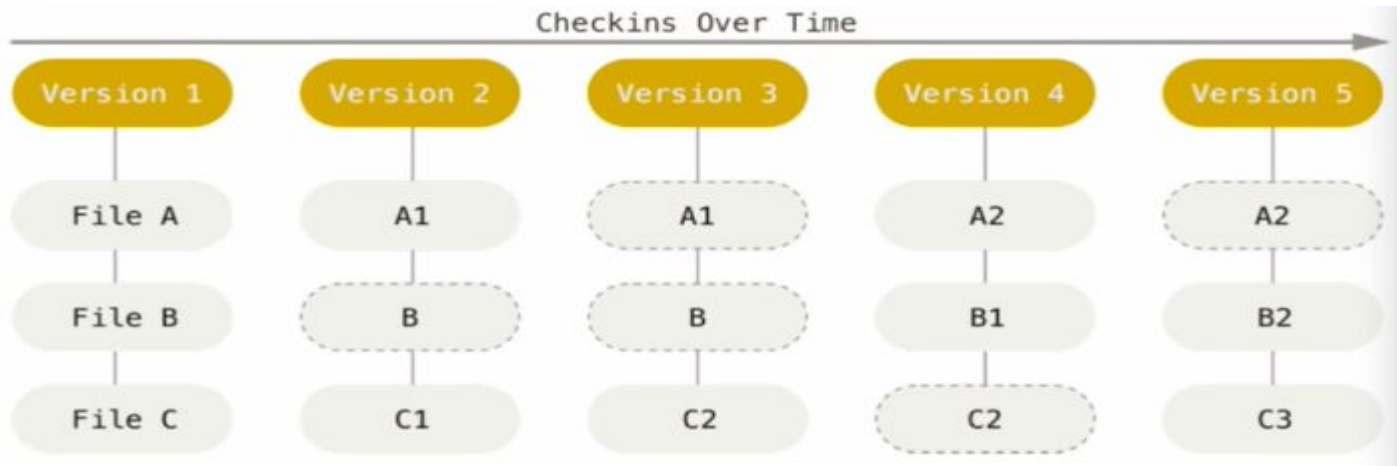
- Un sistema de control de versiones distribuido trabaja con el principio de que cada desarrollador “clona” el repositorio del proyecto al disco duro de su dispositivo.





- La diferencia entre git y la mayoría de los otros sistemas de control de versión, es que los otros sistemas almacenan la información como una lista de cambios en los archivos.

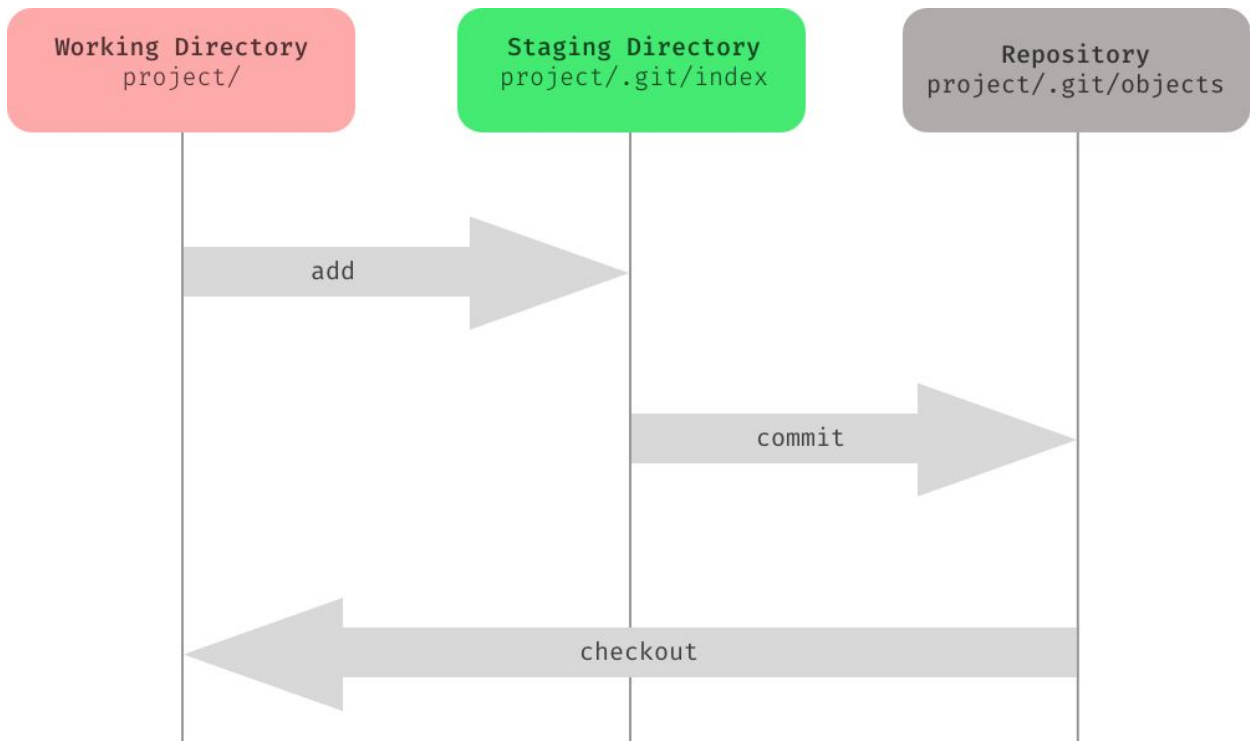


- En cambio, git maneja la información como una serie de “snapshots” o copias instantáneas.
- Cada vez que guardas el estado del proyecto git lo que hace es guardar una copia de tus archivos en ese momento. Para ser eficiente, si un archivo no fue modificado en lugar de copiarlo de nuevo, guarda un enlace a la copia anterior que ya había guardado (nodos punteados).



- 
- La mayoría de las operaciones en GIT son locales (solamente necesitan archivos y recursos locales). Mejora en lo relativo al retardo de red, al ser muchas de las operaciones prácticamente inmediatas.
 - GIT tiene integridad
 - Todo en GIT es verificado mediante una suma de comprobación (checksum) antes de ser almacenado. Además, es identificado a partir de ese momento mediante dicha suma.
 - Es imposible cambiar los contenidos de cualquier archivo o directorio sin que GIT lo sepa.
 - No puedes perder información o sufrir corrupción de archivos sin que GIT sea capaz de detectarlo.
 - El mecanismo de GIT para esta suma de comprobación se conoce como hash SHA-1. Son valores de 40 caracteres hexadecimales que verás en GIT con frecuencia (de hecho GIT guarda todo no por nombre de archivo, sino por el hash de sus contenidos).
 - Cuando realizas acciones en GIT, casi todas ellas añaden información a la base de datos por lo que es difícil (**que no imposible**) que se haga algo que no se pueda enmendar (pero para ello es necesario confirmar copia instantánea y subirlas a repositorios, primer local y luego ya al principal).


- 
- **MUY IMPORTANTE.** Los archivos en GIT se pueden encontrar en 3 estados:
 - **Confirmado (committed).** El archivo está almacenado de forma segura en tu base de datos local.
 - **Modificado (modified).** El archivo ha sido modificado pero no ha sido confirmado a la base de datos.
 - **Preparado (staged).** El archivo modificado ha sido marcado para que vaya a la próxima confirmación.
 - Por tanto, a la hora de trabajar con GIT podemos diferenciar como 3 sitios diferentes:
 - **Directorio de trabajo (Working directory).** Copia de la versión del proyecto que tenemos en un directorio de disco para poder modificarlos o usarlos.
 - **Área de almacenamiento o de preparación (staging area).** Es generalmente una parte del directorio de GIT que tiene información de la próxima confirmación. También es conocido como Index
 - **Repositorio o directorio de GIT (.git directory o repository).** Almacena los metadatos y la base de datos de tu proyecto. Es la parte más importante de GIT y es lo que se copia al clonar un repositorio desde otra computadora (como puede ser github)





Comprender el funcionamiento de las tres zonas de Git es vital para entender su funcionamiento.


- El Directorio de trabajo (Working Directory) es considerado el directorio donde inicializamos git. En esta zona los ficheros están en el estado Untracked (Que no están en seguimiento)
- La zona de preparación (Staging area o Staging Directory) es donde colocamos los ficheros que deseamos tener en seguimiento, cuando los ficheros se encuentran en esta zona, git nos indicará si el fichero es nuevo para él, si un fichero ha sido modificado o si ha sido borrado por completo para poder deshacer esa operación local incluso antes de confirmar los cambios.
- Por último, el repositorio git o zona de commits (Repository) son los cambios confirmados después de las verificaciones a través de la zona de preparación. Estos cambios se aplicarán directamente sobre el directorio de trabajo y el ciclo empezará de nuevo.

- 
- El flujo de trabajo en GIT suele ser el siguiente:
 1. Modificas los archivos en tu directorio de trabajo (previamente por supuesto te has tenido que hacer un clon del proyecto en tu máquina local).
 2. Preparar los archivos a confirmar, añadiéndoles al área de preparación.
 3. Confirmas cambios: Toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera pertinente en tu directorio GIT, es decir, los almacenas en tu copia local (Faltaría por supuesto, la parte en la que se confirma esa copia en el repositorio principal).



Configurando GIT por primera vez


- Es posible personalizar y configurar GIT en tu computadora (una vez por computadora aunque se actualice GIT).
- Para ello se utiliza la herramienta **git config** que permite configurar ciertas variables relacionadas con el aspecto y funcionamiento de Git. Estas variables se almacenarán en tres niveles.
 - Valores para todos los usuarios del sistema y todos sus repositorios. Si pasas la opción **--system** a git config
 - Valores específicos de tu usuario. Pasando la opción **--global**.
 - Valores específicos del directorio de Git del repositorio que estés utilizando actualmente. No se indica ninguna opción
- Cada nivel sobrescribe los valores del nivel anterior. Los valores específicos del repositorio actual tienen preferencia sobre específicos de usuario y estos para los de todos los usuarios.

- 
- Lo primero que configuraremos es nuestro nombre de usuario y dirección de correo electrónico (es importante ya que cuando se hace una confirmación usará esta información)
 - `git config --global user.name "Guillermo Palazón"`
 - `git config --global user.email guillermo.palazon@murciaeduca.es`
 - Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando **sin** la opción `--global` cuando estés en ese proyecto.
 - Si quieres comprobar tu configuración, puedes usar el comando **git config --list**
 - Se puede especificar si quieres solamente la configuración de todos los usuarios (`--system`) o la del usuario específico (`--global`)
 - Puedes pedir también un valor concreto: `git config --global user.name`



Repositorios


- Una vez que ya tenemos un directorio de trabajo si queremos hacer un respaldo del mismo utilizaremos el comando: **git init**
- Lo más lógico es que se haga cuando se inicia un proyecto pero es posible hacerlo en cualquier momento, solamente que las sucesivas modificaciones que se hayan hecho con anterioridad no habrán sido respaldadas ni se tendrá ninguna información sobre ellas.
- Al lanzar un `git init` le estamos diciendo a la herramienta git que cree las estructuras necesarias para albergar un repositorio de GIT.
- Una vez ejecutado, crea GIT crea dos áreas donde se almacenarán los archivos
 - Área de ensayo (staging area). Es un área temporal.
 - Repositorio local. Es donde realmente se almacenan los archivos y de donde se recuperarán.

- 
- Al lanzar un **git init** se van a crear automáticamente unas carpetas en el directorio que se ha lanzado
 - A continuación podemos ver un ejemplo de carpeta vacía antes de lanzarse el git init (utilizamos opción -a que nos muestra archivos y carpetas ocultas)


```
-----
MacBook-Air-de-Guillermo:Proyectos guillermopalazoncano$ cd GitPrueba
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ ls
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ ls -a
.
..
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ █
```

- Al lanzar el git init se crea una nueva carpeta (oculta) en el directorio

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ ls -a
.
..
.git
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ █
```

- 
- No tenemos que preocuparnos de esta carpeta, ya que es GIT el que trabajará sobre la misma
 - La carpeta GitPrueba sería en este momento nuestro Working Directory
 - Al lanzar ese git init nos ha dado una serie de información sobre el repositorio creado que iremos descubriendo su significado a lo largo de la presentación.

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git init
ayuda: Usando 'master' como el nombre de la rama inicial. Este nombre de rama predeterminado
ayuda: está sujeto a cambios. Para configurar el nombre de la rama inicial para usar en todos
ayuda: de sus nuevos repositorios, reprimiendo esta advertencia, llama a:
ayuda:
ayuda: git config --global init.defaultBranch <nombre>
ayuda:
ayuda: Los nombres comúnmente elegidos en lugar de 'master' son 'main', 'trunk' y
ayuda: 'development'. Se puede cambiar el nombre de la rama recién creada mediante este comando:
ayuda:
ayuda: git branch -m <nombre>
```

- 
- Creo un archivo de extensión md (tipo de archivo markdown) al que voy a llamar readme.md por línea de comandos
 - Lo creo que el siguiente comando que va a crear el archivo con una única línea que comienza por # que es similar a un <h1> de HTML y de texto Título del archivo readme.md

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "# Título del archivo readme.md" > readme.md
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ ls
readme.md
```

- Más información sobre el lenguaje Markdown y alguna de sus principales marcas a utilizar a la hora de escribir: <https://markdown.es/>

- 
- Siempre se puede ver el estado en el que se encuentra un Working Directory con **git status**

```
readme.md
```

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status
```

```
En la rama master
```

```
No hay commits todavía
```

```
Archivos sin seguimiento:
```

```
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
```

```
    readme.md
```

```
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ █
```

- Indica que no tenemos ningún archivo en seguimiento (staging area) y que no hay commits todavía. Además nos indica ya el comando que deberíamos realizar para hacer un seguimiento (pasar el archivo al staging area) que es **git add**

- Si hago el `git add readme.md` el fichero pasa ahora al área de revisión

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git add readme.md
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status
En la rama master

No hay commits todavía

Cambios a ser confirmados:
  (usa "git rm --cached <archivo>..." para sacar del área de stage)
    nuevos archivos: readme.md

MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ █
```

- Fijarse que al hacer un `git status` me cambia el color en el nombre del fichero, ha pasado de rojo a verde)
- Ese fichero aún no está en mi repositorio local, si quiero hacer una revisión tengo que utilizar el fichero `git commit` al que se le puede indicar un texto para la misma (opción `-m`)



```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git commit -m "Primera versión. Se introduce el fichero readme.md"
[master (commit-raíz) 7a2bc3c] Primera versión. Se introduce el fichero readme.md
 1 file changed, 1 insertion(+)
 create mode 100644 readme.md
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$
```


- Ya tengo mi primera foto (instantánea) en mi repositorio local
- Estoy en la rama por defecto y ya no hay nada que aprobar: el directorio de trabajo está limpio ya que no tenemos ningún fichero modificado en el directorio de trabajo y tampoco tenemos nada pendiente de hacer commit en el directorio de preparación.
- Si utilizo el comando **git log --oneline** podemos ver las revisiones y podemos ver su HASH ID (el identificador de la versión)

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git log --oneline
7a2bc3c (HEAD -> master) Primera versión. Se introduce el fichero readme.md
```


- A continuación voy a crear 3 ficheros con la misma extensión y tres ficheros con extensión tmp

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "# Fichero para hacer pruebas" > fichero1.md
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "# Fichero para hacer pruebas" > fichero2.md
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "# Fichero para hacer pruebas" > fichero3.md
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "Prueba" > fichero1.tmp
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "Prueba" > fichero2.tmp
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ echo "Prueba" > fichero3.tmp

MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ ls -l
total 56
-rw-r--r--@ 1 guillermopalazoncano  staff  29 23 ene 18:50 fichero1.md
-rw-r--r--  1 guillermopalazoncano  staff   7 23 ene 18:57 fichero1.tmp
-rw-r--r--  1 guillermopalazoncano  staff  29 23 ene 18:50 fichero2.md
-rw-r--r--  1 guillermopalazoncano  staff   7 23 ene 18:57 fichero2.tmp
-rw-r--r--  1 guillermopalazoncano  staff  29 23 ene 18:50 fichero3.md
-rw-r--r--  1 guillermopalazoncano  staff   7 23 ene 18:57 fichero3.tmp
-rw-r--r--  1 guillermopalazoncano  staff  32 23 ene 12:51 readme.md
```

- 
- Si hago un git status me dirá que todos estos ficheros están pendientes de incorporarse al área de seguimiento.

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status
En la rama master
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    .DS_Store
    fichero1.md
    fichero1.tmp
    fichero2.md
    fichero2.tmp
    fichero3.md
    fichero3.tmp
```

- Es posible que los ficheros .tmp que hemos creado NO queramos nunca incluirlos a nuestras copias del repositorio, así como la carpeta oculta .DS_STORE que en ocasiones puede crearse en sistemas operativos MacOS (cada sistema operativo puede tener las suyas)
- **Para indicar a GIT qué ficheros NO queremos que incluya en las copias crearemos el fichero .gitignore** → "nano .gitignore" con el siguiente contenido que indica que no queremos que nos muestre ningún fichero tmp y tampoco el .DS_STORE

```
*.tmp  
.DS_STORE
```

- Vuelvo a hacer un git status y ahora estos ficheros no aparecen

```
MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status  
En la rama master  
Archivos sin seguimiento:  
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)  
  .gitignore  
  fichero1.md  
  fichero2.md  
  fichero3.md
```

- Vamos a añadir todos los ficheros al área staging con un “git add .”, luego haremos un commit y finalmente veremos que existe una nueva versión.

```
[MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git add .
```

```
[MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status
```

```
En la rama master
```

```
Cambios a ser confirmados:
```

```
(usa "git restore --staged <archivo>..." para sacar del área de stage)
```

```
nuevos archivos: .gitignore
```

```
nuevos archivos: fichero1.md
```

```
nuevos archivos: fichero2.md
```

```
nuevos archivos: fichero3.md
```

```
[MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git commit -m "2 versión con 3 ficheros md y el fichero ignore"
```

```
[master 3220138] 2 versión con 3 ficheros md y el fichero ignore
```

```
4 files changed, 5 insertions(+)
```

```
create mode 100644 .gitignore
```

```
create mode 100644 fichero1.md
```

```
create mode 100644 fichero2.md
```

```
create mode 100644 fichero3.md
```

```
[MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git status
```

```
En la rama master
```

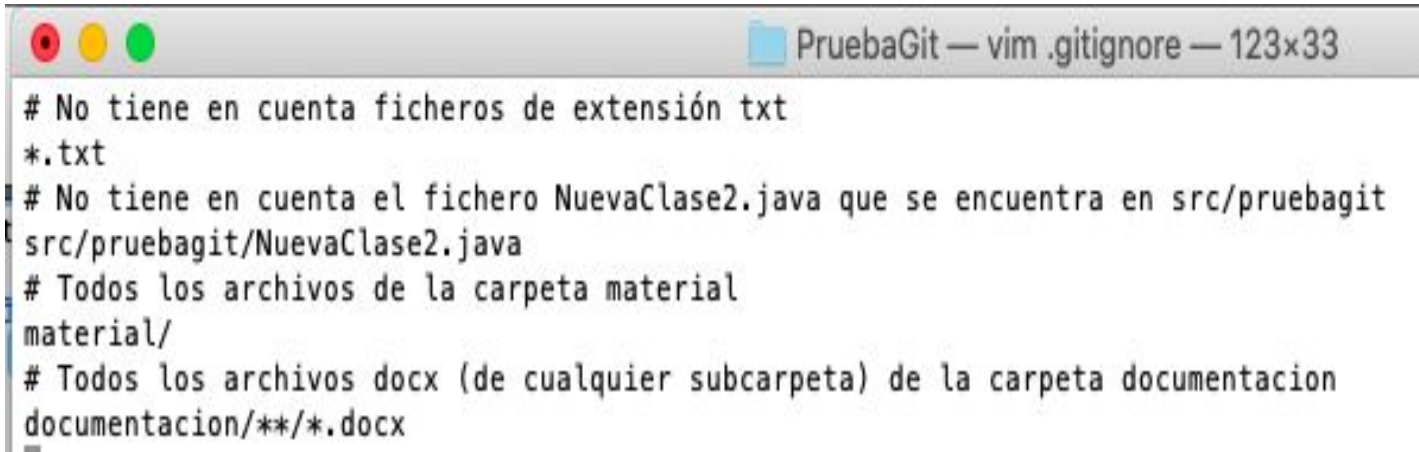
```
nada para hacer commit, el árbol de trabajo está limpio
```

```
[MacBook-Air-de-Guillermo:GitPrueba guillermopalazoncano$ git log --oneline
```

```
3220138 (HEAD -> master) 2 versión con 3 ficheros md y el fichero ignore
```

```
7a2bc3c Primera versión. Se introduce el fichero readme.md
```

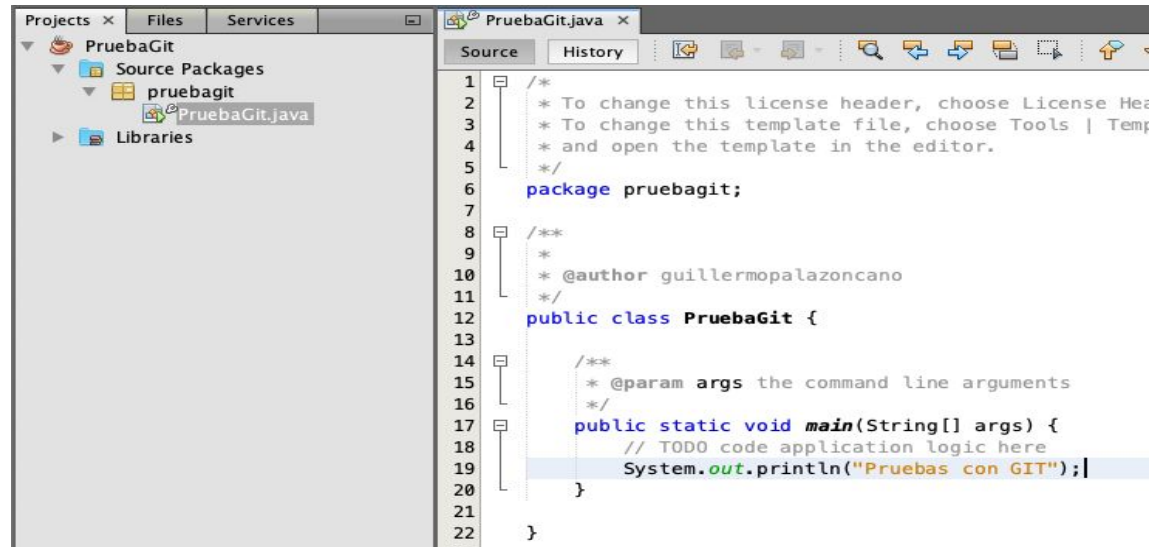
- En el fichero .gitignore se pueden añadir líneas en blanco y también comentarios con el #
- Ejemplos en el .gitignore

A screenshot of a vim editor window. The title bar at the top reads "PruebaGit — vim .gitignore — 123x33". The editor displays the following text:

```
# No tiene en cuenta ficheros de extensión txt
*.txt
# No tiene en cuenta el fichero NuevaClase2.java que se encuentra en src/pruebagit
src/pruebagit/NuevaClase2.java
# Todos los archivos de la carpeta material
material/
# Todos los archivos docx (de cualquier subcarpeta) de la carpeta documentacion
documentacion/**/*.docx
```

Repositorios e IDE (Netbeans)

- Vamos a crear un proyecto en NetBeans cualquiera, en este caso lo he llamado PruebaGit con una sola línea de mensaje en terminal



The screenshot shows the NetBeans IDE interface. On the left, the 'Projects' pane displays a project named 'PruebaGit' with a sub-package 'pruebagit' containing the file 'PruebaGit.java'. The main editor window shows the source code of 'PruebaGit.java' with the following content:

```
1  /*
2  * To change this license header, choose License Headers from Project
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package pruebagit;
7
8  /**
9   *
10  * @author guillermopalazoncano
11  */
12  public class PruebaGit {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          System.out.println("Pruebas con GIT");
20      }
21  }
```

- 
- Nos situamos en la carpeta en la que se encuentra el proyecto y lanzaremos el git init

```
-----
MacBook-Air-de-Guillermo:~ guillermopalazoncano$ cd PruebaGit
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ ls
build.xml      manifest.mf    nbproject      src
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git init
Inicializado repositorio Git vacío en /Users/guillermopalazoncano/PruebaGit/.git/
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ ls
build.xml      manifest.mf    nbproject      src
```

- Si deseas empezar a controlar versiones de archivos existentes, debes comenzar el seguimiento de esos archivos y hacer una confirmación inicial.
- Puedes conseguirlo con unos pocos comandos git add para especificar qué archivos quieres controlar, seguidos de un git commit para confirmar los cambios:

- Este comando puede ser usado para agregar archivos al index. Por ejemplo, el siguiente comando agrega todos los archivos java (.java) en el directorio local del index: `git add *.java` (incluirá los de subcarpetas incluido).

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git add *.java
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
```

No hay commits todavía

Cambios a ser confirmados:

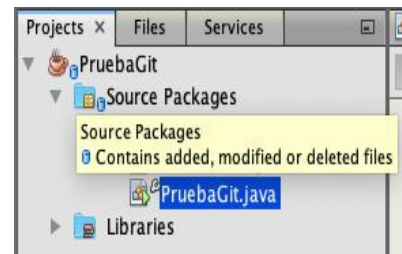
(usa "`git rm --cached <archivo>...`" para sacar del área de stage)

nuevos archivos: `src/pruebagit/PruebaGit.java`

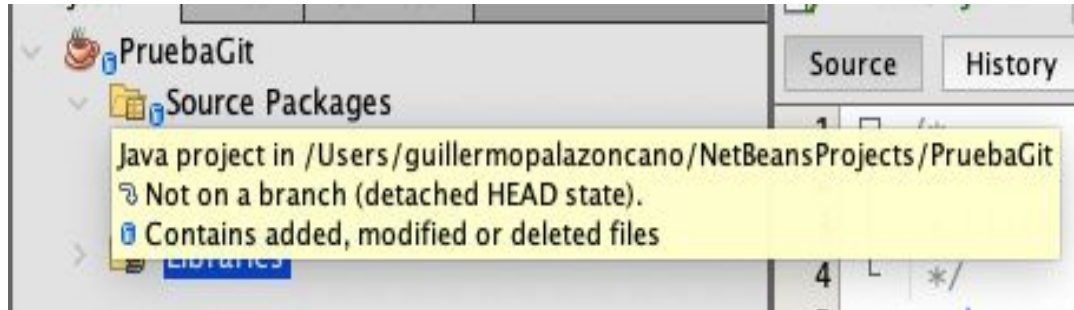
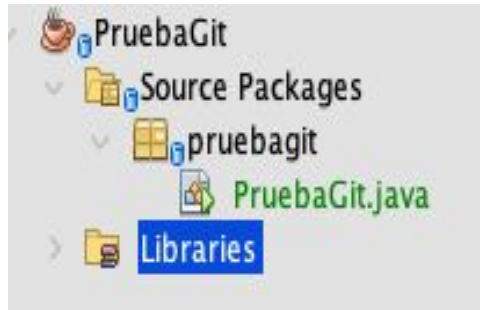
Archivos sin seguimiento:

(usa "`git add <archivo>...`" para incluirlo a lo que se será confirmado)

`build.xml`
`manifest.mf`
`nbproject/`



- Si volvemos a nuestro proyecto en Netbeans vemos que detecta que el proyecto está añadido a GIT




- Añado una nueva clase al proyecto (NuevaClase.java) y a continuación vamos a subir todos los archivos al index, para ello se hace **git add**.

```
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git add .  
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status  
En la rama master
```

No hay commits todavía

Cambios a ser confirmados:

```
(usa "git rm --cached <archivo>..." para sacar del área de stage)  
nuevos archivos: build.xml  
nuevos archivos: manifest.mf  
nuevos archivos: nbproject/build-impl.xml  
nuevos archivos: nbproject/genfiles.properties  
nuevos archivos: nbproject/private/private.properties  
nuevos archivos: nbproject/project.properties  
nuevos archivos: nbproject/project.xml  
nuevos archivos: src/pruebagit/NuevaClase.java  
nuevos archivos: src/pruebagit/PruebaGit.java
```

- 
- Podemos ver en la imagen que donde nos indica los cambios a ser confirmados indica el comando con el que podríamos sacar un archivo del index (del área stage): **git rm --cached <archivo>**
 - Probar a sacar el fichero último que habíamos creado (NuevaClase.java).
 - Posteriormente lanzamos nuevamente un git status y nos indicará que ese fichero (NuevaClase.java) está sin seguimiento y nos vuelve a recordar la orden que debemos introducir para volver a incorporarlo.



```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git rm --cached src/pruebagit/NuevaClase.java
rm 'src/pruebagit/NuevaClase.java'
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
```

No hay commits todavía

Cambios a ser confirmados:


(usa "git rm --cached <archivo>..." para sacar del área de stage)

```
nuevos archivos: build.xml
nuevos archivos: manifest.mf
nuevos archivos: nbproject/build-impl.xml
nuevos archivos: nbproject/genfiles.properties
nuevos archivos: nbproject/private/private.properties
nuevos archivos: nbproject/project.properties
nuevos archivos: nbproject/project.xml
nuevos archivos: src/pruebagit/PruebaGit.java
```

Archivos sin seguimiento:

(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)

```
src/pruebagit/NuevaClase.java
```

- 
- Una vez que ya tenemos decididos que ficheros queremos confirmar podemos ejecutar el comando correspondiente git commit

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git commit -m 'Subida inicial al repositorio'
[master (commit-raíz) c8cf523] Subida inicial al repositorio
 8 files changed, 1988 insertions(+)
 create mode 100644 build.xml
 create mode 100644 manifest.mf
 create mode 100644 nbproject/build-impl.xml
 create mode 100644 nbproject/genfiles.properties
 create mode 100644 nbproject/private/private.properties
 create mode 100644 nbproject/project.properties
 create mode 100644 nbproject/project.xml
 create mode 100644 src/pruebagit/PruebaGit.java
```

- Si realizamos un git status vemos ahora que únicamente nos indica el archivo que no tenemos en seguimiento.

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    src/pruebagit/NuevaClase.java
```

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)

- Podemos ver ahora en NetBeans que PruebaGit.java aparece en estado Negro por estar ya confirmado, mientras que NuevaClase.java aparece en verde.





Clonando un repositorio

- Si deseo obtener una copia de un repositorio existente (un proyecto en el que vas a empezar a trabajar) el comando que se utiliza es **git clone url_repositorio**.
- Este comando es similar al checkout de otros sistemas de control de versiones como puede ser Subversion.
- Realmente hay diferencia, puesto que cuando haces clone de un repositorio se recibe una copia de casi todos los datos que se tiene del mismo en el servidor (esta copia incluso podría utilizarse en el caso de que el servidor perdiera el mismo por algún problema)
- Con el comando “**git clone https://abcde.com/abcde/abcde carpetaEnMiMaquina**” se descargara el repositorio de la URL en la carpeta carpetaEnMiMaquina



Estado archivos en GIT

- He creado un nuevo archivo en el proyecto que se llama NuevoFichero.java y lo he incorporado al área de seguimiento con un `git add nombre_fichero`
- He añadido una nueva línea en el fichero PruebaGit.java definiendo una variable entera dentro del método main a la que le he dado un valor (se podría insertar cualquier cosa)
- Si hacemos un `git status` nos indicará que en la actualidad tenemos tres archivos cada uno de manera diferente respecto a Git
 - NuevoFichero.java está en el área stage pendiente de confirmarse (pendiente de commit)
 - PruebaGit.java está modificado. Es un archivo de los que ha sido confirmado en alguna ocasión, pero sus últimos cambios no han sido confirmados.
 - NuevaClase.java es un archivo sin seguimiento (ni siquiera está porque hicimos el `git rm` en el área de seguimiento).



[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano\$ git status

En la rama master

Cambios a ser confirmados:

(usa "git restore --staged <archivo>..." para sacar del área de stage)

nuevos archivos: src/pruebagit/NuevoFichero.java

Cambios no rastreados para el commit:

(usa "git add <archivo>..." para actualizar lo que será confirmado)

(usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)

modificados: src/pruebagit/PruebaGit.java

Archivos sin seguimiento:

(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)

src/pruebagit/NuevaClase.java



- Aunque en Netbeans tanto NuevaClase.java como NuevoFichero.java están en color verde, si colocamos el cursor encima de ellos el mensaje que muestra no es el mismo, aunque también muy similar y nos puede dar pie a error.
- Pruebagit.java está en color azul de encontrarse modificado.
- RECOMENDACIÓN → Utilizar Git Status

- Vamos a comenzar a rastrear el archivo NuevaClase.java

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git add src/pruebagit/NuevaClase.java
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
    nuevos archivos: src/pruebagit/NuevaClase.java
    nuevos archivos: src/pruebagit/NuevoFichero.java

Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificados:      src/pruebagit/PruebaGit.java
```

- Sin hacer el commit, ahora añado un nuevo método a NuevaClase.java.
- A continuación si hago un git status podemos ver que es verdad que tiene cambios a ser confirmados en NuevaClase.java pero a su vez me dice que existen cambios no rastreados en el commit (IMPORTANTE: si hago un commit se subirá la versión de NuevaClase.java) anterior a añadir el nuevo método, es decir, la versión que existía al realizarse el git add)

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
```

```
En la rama master
```

```
Cambios a ser confirmados:
```

```
(usa "git restore --staged <archivo>..." para sacar del área de stage)
```

```
nuevos archivos: src/pruebagit/NuevaClase.java
```

```
nuevos archivos: src/pruebagit/NuevoFichero.java
```


```
Cambios no rastreados para el commit:
```

```
(usa "git add <archivo>..." para actualizar lo que será confirmado)
```

```
(usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
```

```
modificados:      src/pruebagit/NuevaClase.java
```

```
modificados:      src/pruebagit/PruebaGit.java
```

- 
- Si en Netbeans con el cursor consulto el estado de NuevaClase.java me indica que está añadido y Modificado.
 - A continuación, hago un commit que confirmará tanto NuevaClase.java como NuevoFichero.java y posteriormente podemos ver con git status que NuevaClase.java seguirá estando en cambios no rastreados para el commit (en Netbeans aparecerá ahora en azul)

```
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git commit -m "Subida NuevaClase.java y NuevoFichero.java"
[master f45e8fe] Subida NuevaClase.java y NuevoFichero.java
 2 files changed, 27 insertions(+)
 create mode 100644 src/pruebagit/NuevaClase.java
 create mode 100644 src/pruebagit/NuevoFichero.java
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
Cambios no rastreados para el commit:
(usa "git add <archivo>..." para actualizar lo que será confirmado)
(usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificados:      src/pruebagit/NuevaClase.java
    modificados:      src/pruebagit/PruebaGit.java
```

- Existen versiones de `git status` que ofrecen la información más resumida.
- Estas opciones son **`git status -s`** o **`git status --short`**. Esto nos mostrará los archivos con 3 indicadores
 - `??` → No están rastreados
 - `A` → Preparados (se encuentran en el index o área de preparación)
 - `M` → Están modificados
- En nuestro ejemplo en la actualidad solamente tenemos ficheros que se encuentran en estado modificado

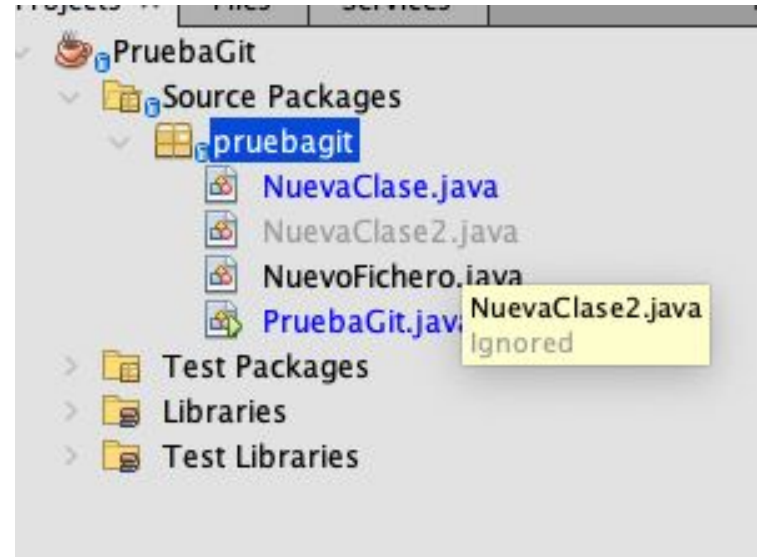
```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status -s
M src/pruebagit/NuevaClase.java
M src/pruebagit/PruebaGit.java
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status --short
M src/pruebagit/NuevaClase.java
M src/pruebagit/PruebaGit.java
```



Ignorar archivos

- Como ya se ha comentado, en git podemos crear un fichero .gitignore para darle una serie de reglas sobre qué archivos no quiero que añada automáticamente (o incluso que no aparezca como no rastreado).
- Esto puede ser útil para archivos que generan automáticamente los sistemas de compilación o incluso archivos temporales que pueden generar algunos de los editores que son utilizados (muchas veces son el mismo nombre del archivo pero finalizado con el carácter tilde (~))

- Creamos una nueva clase NuevaClase2.java, pero no queremos que forme parte de GIT por lo que lo indicaremos en el .gitignore
- Si hacemos un git status no tendremos rastro de NuevaClase2.java
- Además, en el propio NetBeans, nos indica que ese fichero está ignorado (puede tardar en actualizarse)





Ver cambios realizados

- Cuando tienes un fichero modificado o en el área de preparación (index), el comando `git status` únicamente te indica este hecho, pero si queremos saber qué ha cambiado pero aún no has incorporado al área de preparación o incluso si está allí y quieres saber los cambios respecto a su contenido en el repositorio, se debe utilizar el comando **git diff**
- `git diff` es un comando muy potente con diferentes opciones. De manera básica nos indican las líneas que han sido añadidas o eliminadas
 - `git diff` (sin parámetros) nos mostrará los cambios que aún no están preparados.
 - `git diff staged` nos mostrará los que se ha preparado y será incluido en la próxima confirmación


```

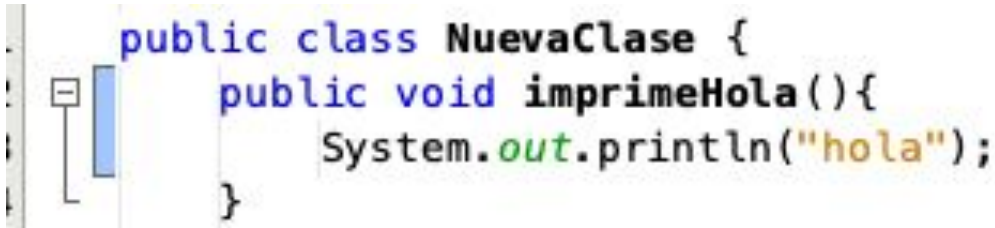
sin cambios agregados al commit (usa git add y/o git commit -a /
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git diff
diff --git a/src/pruebagit/NuevaClase.java b/src/pruebagit/NuevaClase.java
index 0ddb229..92cce56 100644
--- a/src/pruebagit/NuevaClase.java
+++ b/src/pruebagit/NuevaClase.java
@@ -9,5 +9,7 @@ package pruebagit;
 * @author guillermopalazoncano
 */
public class NuevaClase {
-
+   public void imprimeHola(){
+       System.out.println("hola");
+   }
}
diff --git a/src/pruebagit/PruebaGit.java b/src/pruebagit/PruebaGit.java
index 5d07f77..7629455 100644
--- a/src/pruebagit/PruebaGit.java
+++ b/src/pruebagit/PruebaGit.java
@@ -15,6 +15,8 @@ public class PruebaGit {
    */
    public static void main(String[] args) {
        // TODO code application logic here
+       int numero = 5;
+       System.out.println("Pruebas con GIT");
    }
}
}

```


- Si ejecutamos `git diff --staged` no mostrará nada al no tener ningún fichero en dicha área. Vamos a pasar con `git add` a `NuevaClase.java` al staged (área de preparación o index). Posteriormente el `git diff --staged` ya nos indicará los cambios respecto al del repositorio.

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git diff --staged
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git add src/pruebagit/NuevaClase.java
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git diff --staged
diff --git a/src/pruebagit/NuevaClase.java b/src/pruebagit/NuevaClase.java
index 0ddb229..92cce56 100644
--- a/src/pruebagit/NuevaClase.java
+++ b/src/pruebagit/NuevaClase.java
@@ -9,5 +9,7 @@ package pruebagit;
 * @author guillermopalazoncano
 */
public class NuevaClase {
-
+    public void imprimeHola(){
+        System.out.println("hola");
+    }
}
```


- 
- Esto se puede verificar también en NetBeans. Si nos metemos dentro de la clase podemos ver cierto código que se encuentra en color azul como que está modificado (también tendríamos código en color verde en el caso correspondiente).



```
public class NuevaClase {  
    public void imprimirHola(){  
        System.out.println("hola");  
    }  
}
```

- 
- Si a la hora de hacer el commit queremos tener un poco más de información sobre lo que vamos a confirmar, podemos hacer un git commit (sin la opción -m y el comentario de la subida)
 - Esto nos abrirá nuestro editor con una línea en blanco en la parte superior en la que se pondrá el comentario y cuando se añada esta línea y se guarde directamente hará el commit correspondiente

```
# Por favor ingresa el mensaje del commit para tus cambios. Las
# líneas que comiencen con '#' serán ignoradas, y un mensaje
# vacío aborta el commit.
#
# En la rama master
# Cambios a ser confirmados:
#     modificados:      src/pruebagit/NuevaClase.java
#
# Cambios no rastreados para el commit:
#     modificados:      src/pruebagit/PruebaGit.java
#
# Archivos sin seguimiento:
#     .gitignore
#
```

- 
- Si queremos saltarnos la opción del área de preparación (index) y querer confirmar todos los archivos que se encuentran modificados , se puede añadir al git commit la opción -a
 - Si hacemos un git status podemos ver que tenemos por ejemplo Pruebagit.java modificado pero que no se encuentra en el área de preparación, si hacemos el git commit con la opción -a directamente se subirá su confirmación
 - Esta opción es interesante si a continuación del git add vas a realizar el git commit.
 - Si quieres poner el mensaje tendrás que indicar tanto la opción -a como -m

```
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
      modificados:      src/pruebagit/PruebaGit.java

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
      .gitignore

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git commit -a -m "Subida de prueba saltándonos la subida al index"
[master 6dd59c6] Subida de prueba saltándonos la subida al index
 1 file changed, 2 insertions(+)
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
[En la rama master
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
      .gitignore

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$
```



Eliminar archivos

- Si se ha eliminado un fichero del proyecto (en nuestro caso vamos a eliminar NuevoFichero.java) si hacemos un git status este nos aparece como un cambio a ser confirmado.

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
En la rama master
Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
    borrados:      src/pruebagit/NuevoFichero.java

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    .gitignore
```



- 
- Podemos por tanto hacer dicha confirmación


```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git commit -m "Borrado del fichero  
NuevoFichero.java"  
[master 1cddc24] Borrado del fichero NuevoFichero.java  
1 file changed, 14 deletions(-)  
delete mode 100644 src/pruebagit/NuevoFichero.java
```

Recuperar versiones

- Precisamente una de las posibilidades de trabajar con un gestor de versiones es poder volver a versiones anteriores de un fichero en caso de desearlo.
- En el caso de que se hayan realizado cambios en un fichero que ya tengas versionado en el repositorio y quieras deshacer los mismos para quedarte con la última versión del mismo podemos conseguirlo mediante el comando **git restore <filename>** (también podría haber sido git restore HEAD filename). Anterior a git restore se ha utilizado (y se sigue utilizando) git checkout
- En el fichero NuevaClase.java voy a realizar un cambio, por ejemplo, añadiendo una nueva línea al método. Si realizo un git status estará en estado modificado.


```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status --short
 M src/pruebagit/NuevaClase.java
?? .gitignore
```

- 
- git restore se puede usar en tres situaciones diferentes, dependiendo de la situación actual, si nos gusta revertir el trabajo (cambiar el contenido del fichero) en la copia de trabajo o en el índice, o en ambos. Para ellos se utilizarán o no las opciones --staged y --worktree, junto con la opción --source que nos permitirá indicar la versión desde la que queremos hacer la copia
 - git restore [--worktree] <file> → Sobrescribirá únicamente la copia del trabajo pero no la del index (se puede omitir la opción worktree al ser la opción por defecto)
 - git restore --staged <file> → Sobreescibirá nuestro archivo de índice con el presente HEAD del repositorio local
 - git restore --staged --worktree --source HEAD <file> → Sobreescibirá tanto el archivo índice como la copia del trabajo.

- 
- Quiero deshacer esos cambios que he hecho en el NuevaClase.java y traer la última versión que se tiene almacenada en el repositorio

```
git restore --staged --worktree --source HEAD src/pruebagit/NuevaClase.java
```

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status --short
?? .gitignore
```

- 
- En esta misma clase vamos a añadir un nuevo método y vamos a incorporarlo al repositorio (una vez creado el método podemos hacer el git add y el git commit o directamente el git commit con la opción -a)

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git commit -a -m "Nuevo método en la clase NuevaClase.java"
[master 9510845] Nuevo método en la clase NuevaClase.java
1 file changed, 4 insertions(+)
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status --short
?? .gitignore
```

- Con el comando **git log nombre_fichero**, podemos ver todas las versiones en las que ha participado. Con **git log --stat** podemos ver todas las versiones con mucha información y con **git log --name-status** los ficheros modificados en cada versión (existe una amplia variedad de opciones de git log)

- En nuestro caso podemos ver que tenemos versiones del repositorio en los que dicha clase se ha visto modificada.

```
src/pruebagit/Pruebagit.java
[MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git log src/pruebagit/NuevaClase.java
commit 95108457583d9893a5cbdd5b67e2903846251f90 (HEAD -> master)
Author: Guillermo Palazón <guillermo.palazon@murciaeduca.es>
Date: Mon May 9 15:20:56 2022 +0200
```

Nuevo método en la clase NuevaClase.java

```
commit 25b2e85399fd254b854615440d60a9a7df2f260b
Author: Guillermo Palazón <guillermo.palazon@murciaeduca.es>
Date: Mon May 9 13:21:00 2022 +0200
```

Nueva versión con las modificaciones de NuevaClase.java

```
commit f45e8feaf805b15ab0039b591f53c731ae84268e
Author: Guillermo Palazón <guillermo.palazon@murciaeduca.es>
Date: Mon May 9 12:21:16 2022 +0200
```

Subida NuevaClase.java y NuevoFichero.java

- En el caso de querer volver a tener el fichero como en una versión anterior a la hora de hacer el checkout deberemos indicar el código de dicha versión.
- Por ejemplo, queremos volver al código que tenía en su primera versión, es decir, cuando hicimos la primera subida de la clase al repositorio, es decir, la versión que en nuestra imagen (en la vuestra por supuesto será diferente) comienza por f45.
- Si hacemos un git status este fichero se encontrará en estado modificado

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git restore --worktree --staged --source f45e8fe  
af805b15ab0039b591f53c731ae84268e src/pruebagit/NuevaClase.java
```

```
MacBook-Air-de-Guillermo:PruebaGit guillermopalazoncano$ git status
```

En la rama master

Cambios a ser confirmados:


(usa "git restore --staged <archivo>..." para sacar del área de stage)

modificados: src/pruebagit/NuevaClase.java

Archivos sin seguimiento:

(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)

.gitignore

- 
- Ahora la versión antigua del archivo ha sido restaurada en el directorio de trabajo y en el index. No olvides que una vez restaurado el archivo, es necesario volver a añadir el archivo y volver a hacer un «commit» ya que el archivo cambió.
 - También podemos llevar no sólo un archivo a un punto predeterminado, si no todos los archivos del repositorio, para ello solamente debemos omitir nombre de archivo (también podemos escribir: `git checkout version`)
 - OJO: Al hacer esto, tu repositorio va atrás en el tiempo por completo, así que se podría perder trabajo → Por ello, en GIT siempre se recomienda trabajar con ramas.