

APUNTES DE GIT

https://www.programacionfacil.org/cursos/git_github

Git es un software de control de versiones para desarrolladores **VCS** en inglés Version control system.

GitHub es una plataforma que mantiene repositorios de código en internet para que podamos trabajar con un equipo de personas en un mismo proyecto. En resumen **Git** es una herramienta que permite entre otras cosas manejar **GitHub**. A groso modo, **Git** trabaja en local (en nuestro propio equipo) y **GitHub** en la nube.

Instalación

Descargar GIT <https://git-scm.com/>

- Next
- Marcar Desktop icon
- Editor por defecto: Visual Studio Code, Eclipse, Atom, Neatbeans... en cualquier momento puede cambiarse <https://docs.github.com/es/get-started/getting-started-with-git/associating-text-editors-with-git>
- Let Git decide (así la rama principal siempre tendrá el nombre de máster)
- Git from the command line (recommended) ---- es para poder usar otras consolas ej. PowerShell de Windows
- El resto opciones por defecto
- Install
- Finish



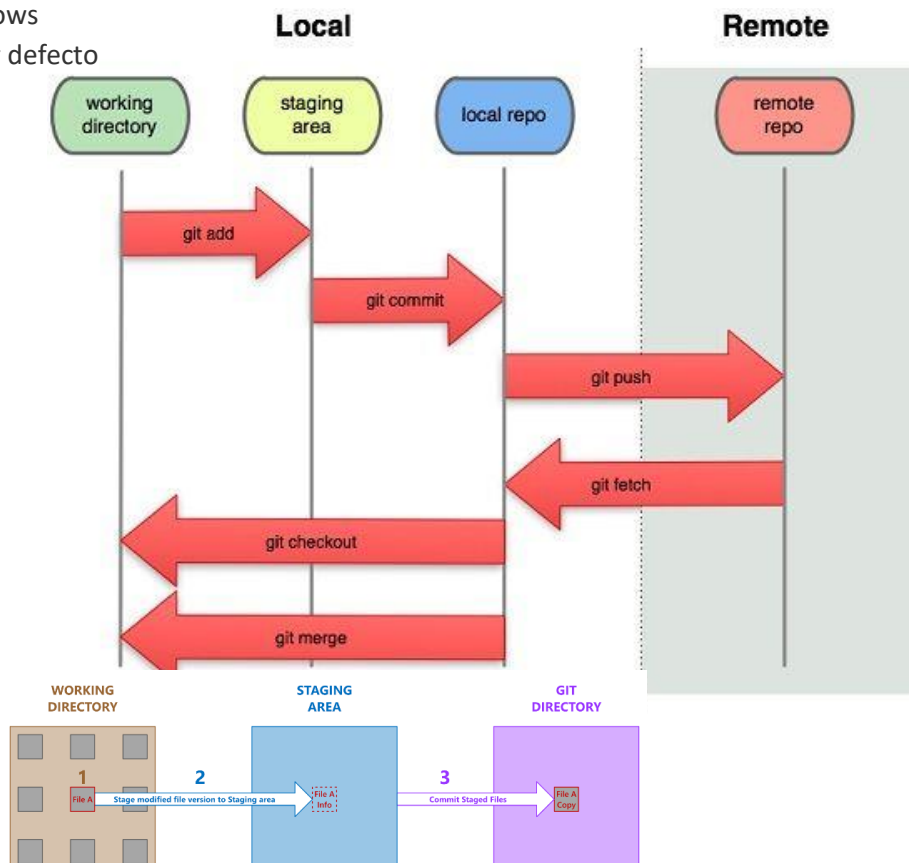
Herramienta de control de versiones distribuida

Herramienta de código abierto que los desarrolladores instalan localmente para gestionar el código fuente



Plataforma basada en la nube

Servicio en línea al que los desarrolladores que utilizan Git pueden conectarse y cargar o descargar recursos



1. Manejo de la consola Git Bash

- Ver la versión que hemos instalado: `git -version`
- El directorio en el que estamos aparece arriba detrás del símbolo de git y la virgulilla lo representa
- Podemos estar varios usuarios trabajando en el mismo proyecto, por eso debemos configurar el nombre del usuario que está trabajando y su mail

```
git config --global user.name "aprendiendo_git"
```

```
git config --global user.email blorentep@iesch.org
```

• OPCIONES DE COMANDO --

`--global` especifica que el usuario y el correo se va a utilizar en todos los proyectos que haga con git. Como vamos a trabajar solos la vamos a usar. Si quisiéramos trabajar en equipo o en solo algunos proyectos quitaríamos el `--global` pero primero deberíamos estar en un directorio de git.

- Limpiar pantalla `clear`
- Ayuda de Git `git help` o `git help -a`
- Salir de la ayuda: `pulsando tecla q`
- En la consola con la `flecha hacia arriba` recuperamos las opciones que hemos ido usando desde que la hemos abierto
- Esta consola `es Case Sensitive`.
- Ayuda por comandos `git comando -help`

Ej. `git add --help` (se abre en nueva pestaña con la ayuda, descripción, ejemplos etc. en el navegador)

- `Exit` cerrar la consola

2. Moverse por el sistema de carpetas y archivos (= Linux)

- Ver dónde estamos ubicados
- `ls` (veo lo que hay en ese directorio ordenado por colorines)
- `Rutas absolutas`:

`cd /c/users/usuario/documents` (cambio de directorio, sitúa el prompt en ese directorio)

`cd ~` (altGr+4) Vuelve al directorio por defecto

`cd ..` (sube un nivel en el directorio)

- `Rutas relativas`:

`cd` (partiendo de la ruta en la que estás puedes bajar de nivel sin especificar toda la ruta)

Ej. `cd documents` desde la raíz por defecto.

- `mkdir` (crea una carpeta ej `mkdir proyecto1`) `cd proyecto1` `mkdir subproyecto1`

3. Crear un repositorio (espacio donde se almacena cierta información)

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~
$ cd documents

Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents
$ mkdir proyectos_git

Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents
$ cd proyectos_git
Usuario@LAPTOP-K7QKBJ9Q MINGW64
~/documents/proyectos_git
$ mkdir proyecto_1

Usuario@LAPTOP-K7QKBJ9Q MINGW64
~/documents/proyectos_git
$ ls
proyecto_1/
```

- Para inicializar un proyecto poner el prompt de inicio dentro de una carpeta:

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git
$ git init proyecto_1
Initialized empty Git repository in
C:/Users/Usuario/Documents/proyectos_git/proyecto_1/.git/
```

- Otra opción entrar directa mente en proyecto_1 y solamente con el comando **git init**
- Vamos a crear un documento html para probar desde Git Bash. (podría ser cualquier otro documento y también crearlo en cualquier otro programa) **nano** se abre archivo, Pego, **Ctrl+ O**, **nombre:** index.html, 9 líneas escritas, **ctrl+X** salir (con ls vemos que está y con nano de nuevo podemos editarlo)
- Ahora podemos abrir el proyecto desde el Visual Studio Code si queremos **File>Open Folder**
- Veamos cómo está nuestro repositorio **git status**

En la primera línea nos especifica que estamos en la rama master (On branch master).

En la siguiente línea nos dice que no hemos realizado ningún "commit" (No commits yet).

La palabra "commit" no tiene realmente una traducción consensuada para Git, puedes ver una discusión sobre la traducción en este enlace. Para que empieces a entender más fácilmente esta palabra, la voy a traducir como confirmar.

Lo siguiente que nos ha devuelto la consola es "Untracked files:". Esto nos indica los archivos que no se encuentran "confirmados" en nuestro repositorio (no les hemos hecho "commit"). Nos aparecen todos los archivos en color rojo, en este caso, solo index.html. Si hubiera más archivos, saldrían aquí.

Finalmente, aparece "nothing added to commit but untracked files present". Esto nos está diciendo que no tenemos nada añadido al área de confirmación (área de staging que ahora veremos).



```
Usuario@LAPTOP-K7QKBJ9Q MINGW64
~/documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master

No commits yet

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

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

¿Qué es el área de staging?

El área de staging (ensayo) es una zona de confirmación donde vamos a ir enviando todos los archivos que creamos convenientes. Una vez hayamos terminado de añadir los que queramos, tendremos que enviar los archivos a commit para que estos sean confirmados en la versión actual de nuestro programa, web, etc. Una vez confirmados, podemos seguir creando y modificando archivos. Cada vez, los deberemos ir pasando por el área de staging para finalmente confirmar los archivos que consideremos como terminados de crear o modificar.

Para que lo entiendas un poco mejor, imagina una persona escribiendo varios archivos de una web. Cuando termina su jornada de trabajo decide salvar los cambios. Para ello, envía los archivos al área de staging. Para simplificarlo, podríamos reemplazar "área de staging" por ¿Está seguro/a de que quiere guardar estos archivos?

Finalmente, este usuario contesta a la pregunta con un "sí" o en Git con un commit.

4. Añadir archivos al área de staging de Git `git add index.html`

Podemos ver si efectivamente se ha añadido el archivo al área de staging con `git status`. Esta vez, nos sigue diciendo que no hemos realizado ningún commit. Sin embargo, aparece en color verde el archivo `index.html` lo cual indica que está listo para ser "confirmado", para hacer el commit. Ya solo nos queda enviar el archivo a **commit** para que forme parte de la primera versión del proyecto de **Git**.

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git add index.html
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory

Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master

No commits yet

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

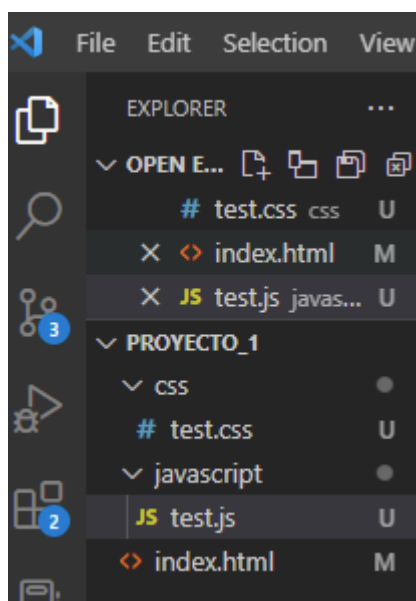
5. Hacer commit en Git `git commit`

Con este comando se hará commit de todos los archivos que hayamos ido guardando en el área de staging. Generaremos un snapshot (instantánea) de estos archivos así tendremos un punto de guardado al que podremos volver en caso de que no nos gusten los cambios o nos hayamos confundido. `git commit -m "texto"`

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git commit -m "creado el index del sitio web v1"
[master (root-commit) 9bb031a] creado el index del sitio web v1
 1 file changed, 9 insertions(+)
 create mode 100644 index.html

Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

6. Añadir más de un archivo al área de staging `git add --all`



Creo un `.css` y un `.javascript` y modifico el `html` con los link a `javascript` y a `css`, podemos ver en una línea verde las modificaciones

Ahora hacemos un `git status` para ver cómo queda, nos muestra las dos carpetas que no tenemos añadidas al proyecto y los cambios en `index.html`

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

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

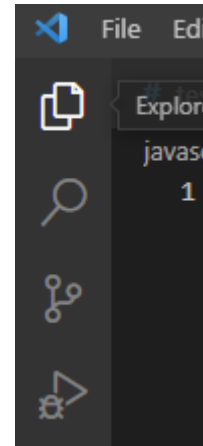
no changes added to commit (use "git add" and/or "git commit -a")
```

Para añadir todo a la vez a Git, tenemos que hacer `git add --all` comprobamos y si está todo bien hacemos un nuevo `git commit -m "añadidas pg css y js y enlazadas con el html v2"`

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git add --all
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory

Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   css/test.css
        modified:   index.html
        new file:   javascript/test.js
```

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git commit -m "añadidas pg css y js y enlazadas con el html v2"
[master 52323e9] añadidas pg css y js y enlazadas con el html v2
3 files changed, 16 insertions(+), 1 deletion(-)
create mode 100644 css/test.css
create mode 100644 javascript/test.js
```



7. Crear ramas (Branch)

Las **ramas** son pequeñas nuevas versiones de tu proyecto.

Por ejemplo tenemos una web creada y podemos crear nuevas ramas para crear modificaciones, en caso de que no nos convenzan podemos ir atrás. Esto con las copias de seguridad es más complejo y es una desventaja. Con Git si en algún momento no nos convence algo, podemos volver atrás sin más, ya sea a la **rama principal** (**master**) o bien a otra **rama** que haya entre medio.

Partimos del proyecto que hemos creado anteriormente. Teniendo en cuenta que ya hemos realizado los primeros commit y que todo está guardado en la rama principal (master).

Para crear una rama lo debes hacer con el siguiente comando: `git branch version_2`

Para ver las ramas que tienes puedes hacerlo con: `git Branch`

El * muestra en la rama que estamos

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git branch
* master
  version_2
```

Para movernos a otra rama debemos hacerlo con `git checkout versión_2` (veremos que ha cambiado el head (parte azul del prompt))

```
Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (master)
$ git checkout version_2
Switched to branch 'version_2'

Usuario@LAPTOP-K7QKBJ9Q MINGW64 ~/documents/proyectos_git/proyecto_1 (version_2)
$
```

hacer modificaciones sobre los archivos html,css, js

Ej

- cargarme `rm test.css`, después eliminar carpetas vacías con `rmdir css` o hacer todo a la vez con `rmdir -r css`
- añadir un párrafo al html
- Luego `git add` y `commit` y si quiero volver a la versión master `git checkout master`

8. Fusionar ramas (Merge) y eliminarlas

La fusión de ramas o **git merge** es la unión de dos ramas. Unión que va a dar como resultado la suma de archivos añadidos y la resta de archivos eliminados. Será una mezcla de las dos ramas..

Vamos a eliminar la rama anterior la de la versión2 para ello tenemos que colocarnos en el master y desde allí hacer **git Branch -D versión_2**

Una vez eliminada, vamos a crear una nueva que implementará nuevas mejoras para la rama master. De paso, veamos **como crear una rama y acceder a la vez a ella con**

git checkout -b rama_imagenes

```
Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git branch -D version_2
Deleted branch version_2 (was 780fdbbc).

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git checkout -b rama_imagenes
Switched to a new branch 'rama_imagenes'
```

Añadimos una carpeta con varias imágenes , las enlace en la página html y elimino la carpeta js con el archivo que lleva dentro. La estructura de archivos del proyecto quedará así:

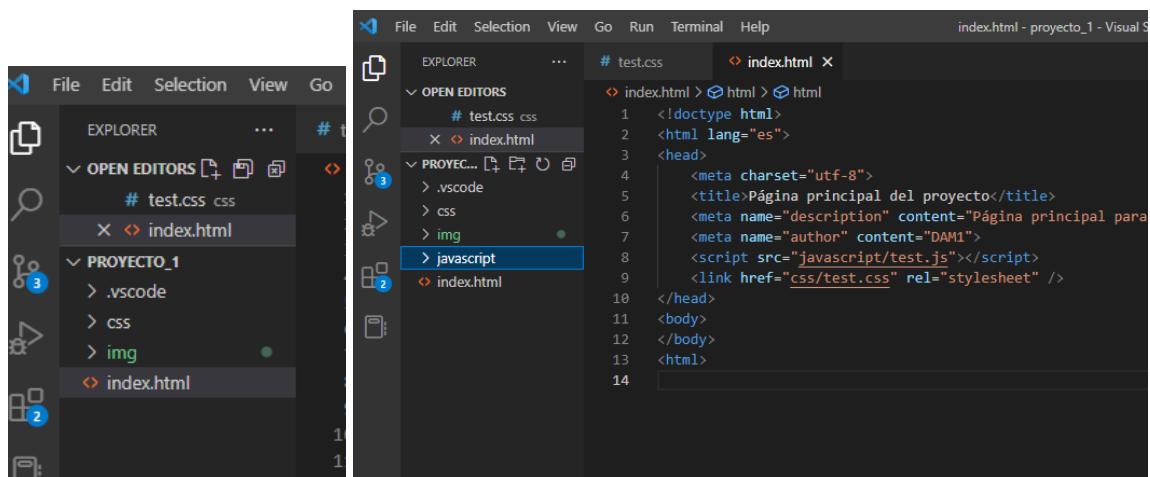
En ocasiones, con pocas modificaciones, puede parecer una tontería enviarlas al área de staging. Cuando tengas la seguridad de que todo está listo para ser confirmado (commit) **puedes hacerlo directamente añadiendo la opción -a.** (OJO ESTO ES PELIGROSO SOLO HACERLO SI ESTÁS SEGURO)

git commit -a -m "Modificadas varias páginas del proyecto"

Veamos las diferencias entre una versión y la otra.

versión rama_imagenes

versión máster (git checkout master)



Ahora realizaremos la fusión sobre la rama máster `git merge rama_imagenes`

```
Usuario@LAPTOP-K7QKB9JQ MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git merge rama_imagenes
Updating 780fdb..98f360b
Fast-forward
 index.html      | 8 +++++--
 javascript/test.js | 1 -
 2 files changed, 6 insertions(+), 3 deletions(-)
 delete mode 100644 javascript/test.js
```

Una vez fusionada ya podemos eliminar la rama_imagenes. En esta ocasión utilizamos -d (esto solo sirve para ramas que ya han sido fusionadas)

`git branch -d rama_imagenes`

9. Ver historial de commits `git log` o `git log --oneline`

```
Usuario@LAPTOP-K7QKB9JQ MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git log --oneline
afc2a52 (HEAD -> master) Corrección error en código html
7a1d9a5 Modificadas varias páginas del proyecto
1e86b8b vuelvo al estado inicial
98f360b Modificadas varias páginas del proyecto
780fdb pongo vscode como editor por defecto
52323e9 añadidas pg css y js y enlazadas con el html v2
9bb031a creado el index del sitio web v1
```

10. Eliminar commits `git reset`

Con `git reset` puedes eliminar commits que no quieras. **Ten en cuenta que esto te será útil si todavía no has subido los commits a un repositorio remoto, por ejemplo, en GitHub.**

Ej si quiero volver al segundo commit haría: `git reset 52323e9`

Si hacemos esto el resto de los commit desaparecen y la información que hemos ido añadiendo también. Te deja en ese punto pero unstaged, es decir habría que hacer un add y un commit

11. Revertir commits `git revert`

Si lo que quieres es no eliminar ningún commit pero no te convencen los últimos cambios lo que puedes hacer es crear un nuevo commit partiendo de uno anterior con `git revert` pero así no perderíamos la información de los otros commit. Es como hacer una copia de un commit anterior para trabajar con ella sin perder nada de lo que has hecho anteriormente

12. Desplazarnos a un commit con `git checkout identificador`

13. Cómo usar Git desde visual Studio Code

<https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-es>

14. ¿Cómo ignorar archivos y carpetas en Git?

En ocasiones, tendremos **cosas que todavía no queramos incluir en el proyecto**, pero que están en la carpeta del mismo. En ese caso, Git nos estará diciendo que tenemos archivos sin añadir al proyecto ("**Untracked**").

Git gestiona los archivos de tres maneras:

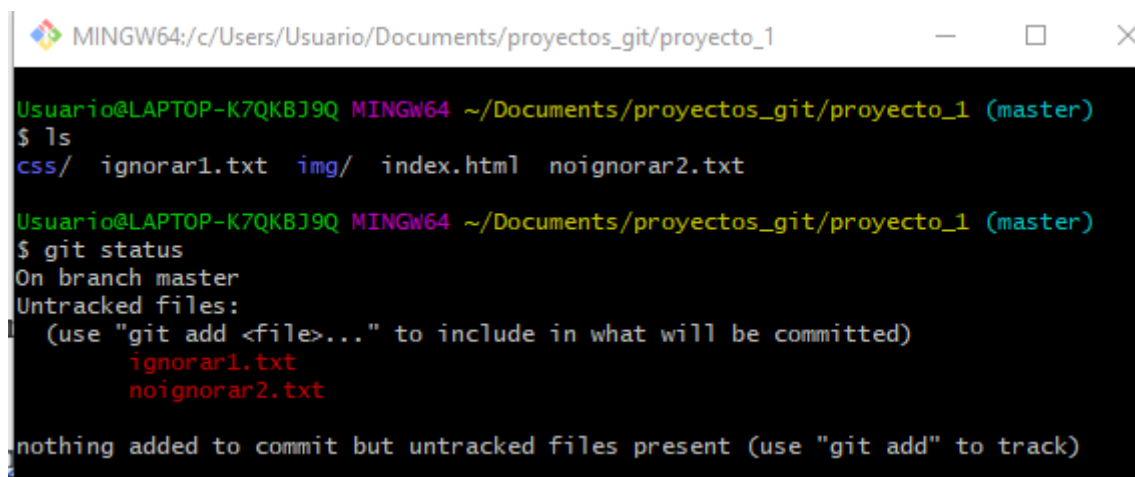
- Sin seguimiento (untracked): no está añadido al repositorio.
- Con seguimiento (tracked): está añadido al repositorio.
- Ignorado (ignored): se ha indicado que lo ignore.

El archivo `.gitignore` para ignorar archivos

El archivo `description.gitignore` es el que nos va a permitir escribir las normas de exclusión.

Podemos excluir tanto archivos sueltos como carpetas.

Creo dos archivos txt uno llamado `ignorar.txt` y el otro `noignorar.txt` dentro de `proyecto_1`



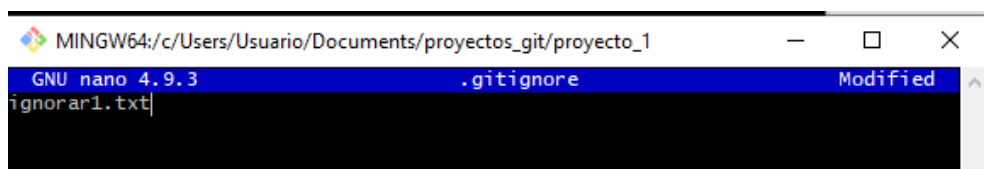
```
MINGW64:/c/Users/Usuario/Documents/proyectos_git/proyecto_1
Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ ls
css/  ignorar1.txt  img/  index.html  noignorar2.txt

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ignorar1.txt
        noignorar2.txt

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

Solo tenemos que crear el archivo **`.gitignore`** y añadirle las reglas que queramos.

¿Cómo crear el archivo `.gitignore`? `nano.gitignore`



```
MINGW64:/c/Users/Usuario/Documents/proyectos_git/proyecto_1
GNU nano 4.9.3 .gitignore Modified
ignorar1.txt|
```

Este archivo (**`.gitignore`**) lo puedes editar hasta con un bloc de notas o cualquier editor de texto.


```

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ nano .gitignore

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        noignorar2.txt

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

```

Para deshacer esto, solo tienes que borrar la línea con el nombre del archivo ignorar1.txt en el .gitignore.

El uso de expresiones regulares en .gitignore

Ignorar todos los archivos con una extensión determinada *.txt (Esto provoca que todos los archivos con extensión .txt sean ignorados.)

```

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

```

Para **excluir una carpeta entera**, lo puedes hacer añadiendo el nombre de la carpeta o ruta y una barra al final `/`.

Ej meto los dos archivos.txt en una carpeta llamada notas_guardadas, modifico el ignore de momento borro todo.

```

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        notas_guardadas/

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

```

MINGW64:/c/Users/Usuario/Documents/proyectos_git/proyecto_1

```

GNU nano 4.9.3          .gitignore
notas_guardadas/

```

```

Usuario@LAPTOP-K7QKB39Q MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

```

Esto provoca que la **carpeta con todos sus archivos sean ignorados**.