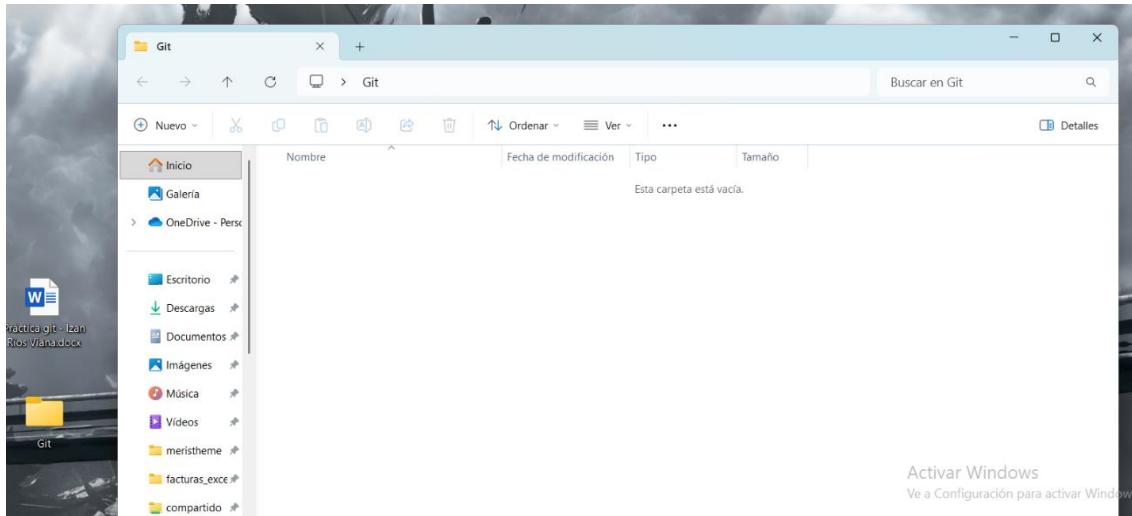


Práctica de Git 1: Fundamentos en una Sola Rama y Gestión de Errores



He creado la carpeta Git en el escritorio.

Fase 1: Preparación del Entorno y Primera Confirmación

Crea un nuevo proyecto:

```
izanr@Izan MINGW64 ~/Desktop/Git
$ git init
Initialized empty Git repository in C:/Users/izanr/Desktop/Git/.git/
Identifícate:
```

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git config --global user.name "Izan"
```

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git config --global user.email "izan.riox.vianax@gmail.com"
```

Primer archivo:

Nombre	Fecha de modificación	Tipo	Tamaño
README.txt	05/10/2025 15:26	Archivo TXT	1 KB

Verifica el estado inicial:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master

No commits yet

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

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

Git detecta el archivo en tu carpeta, pero aún no lo gestiona.

Prepara el archivo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git add README.txt
warning: in the working copy of 'README.txt', LF will be replaced by CRLF the next time Git touches it
```

Verifica el estado preparado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master

No commits yet

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

El archivo está listo para entrar en el próximo commit.

Realiza la primera confirmación:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git commit -m "Primera versión del README"
[master (root-commit) 6dd37ba] Primera versión del README
 1 file changed, 1 insertion(+)
  create mode 100644 README.txt
```

Verifica el estado limpio:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Los cambios están en el repositorio Git (dentro de la carpeta oculta .git), ya forman parte del historial.

Revisa el historial:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git log --oneline
6dd37ba (HEAD -> master) Primera versión del README
```

Fase 2: Modificaciones y Análisis de Diferencias entre Áreas

Modifica el archivo existente:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "Esta es una nueva línea de contenido." >> README.txt
```

```
izanr@Izan MINGW64 ~/Desktop/Git (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:   README.txt

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

Crea un nuevo archivo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "Hola" > documento.txt
```

Observa los cambios no preparados:

```
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:   README.txt

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

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

Compara directorio de trabajo y repositorio (archivo README.txt):

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git diff HEAD -- README.txt
warning: in the working copy of 'README.txt', LF will be replaced by CRLF the next time Git touches it
diff --git a/README.txt b/README.txt
index 2521c45..8763f29 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1,2 @@
 Mi primer proyecto Git
+Esta es una nueva línea de contenido.
```

Esto muestra lo que aún no está guardado en ningún sitio del historial.

Prepara un cambio:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git add README.txt
git status
warning: in the working copy of 'README.txt', LF will be replaced by CRLF the next time Git touches it
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.txt

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

modified: README.txt y documento.txt seguirá como untracked.

Compara directorio de trabajo y área de preparación (README.txt):

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git diff --cached README.txt
diff --git a/README.txt b/README.txt
index 2521c45..8763f29 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1,2 @@
 Mi primer proyecto Git
+Esta es una nueva línea de contenido.
```

Ahora la comparación es lo preparado (staging) contra el repositorio (HEAD). Es exactamente lo que entrará en el próximo commit.

Prepara el segundo archivo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git add documento.txt
git status
warning: in the working copy of 'documento.txt', LF will be replaced by CRLF the
next time Git touches it
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.txt
    new file:   documento.txt
```

Realiza una nueva confirmación:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git commit -m "Añadida nueva línea al README y creado documento.txt"
[master 2d6fa10] Añadida nueva línea al README y creado documento.txt
 2 files changed, 2 insertions(+)
 create mode 100644 documento.txt
```

Revisa el historial actualizado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git log --oneline
2d6fa10 (HEAD -> master) Añadida nueva línea al README y creado documento.txt
6dd37ba Primera versión del README
```

Fase 3: Gestión de Errores y Recuperación ("Eliminar Cagadas")

Escenario A: Descartar cambios en el directorio de trabajo

Introduce un cambio erróneo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "Línea de prueba que no quiero guardar" >> README.txt
```

Verifica el estado:

```
izanr@Izan MINGW64 ~/Desktop/Git (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:   README.txt

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

Deshaz los cambios en el directorio de trabajo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git restore README.txt
```

Los cambios se deshicieron gracias al contenido que Git guarda en el repositorio (último commit). Al usar git restore, Git reemplaza el archivo modificado con la versión confirmada más reciente. Esto me permitió eliminar el error sin tocar el historial de commits, manteniendo el proyecto limpio.

Confirma la eliminación del error:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Escenario B: Despreparar cambios (quitar del staging area)

Modifica un archivo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "Adios" >> documento.txt
```

Prepara el cambio accidentalmente:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git add documento.txt
warning: in the working copy of 'documento.txt', LF will be replaced by CRLF the
next time Git touches it
```

Verifica el estado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   documento.txt
```

Desprepara el archivo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git restore --staged documento.txt
```

Cuando quito el archivo del área de preparación (git restore --staged), los cambios no se pierden, solo salen del “staging”. Siguen existiendo en mi carpeta de trabajo (working directory), por lo que puedo revisarlos antes de confirmarlos. Esto es útil cuando añado algo por error y todavía no quiero hacer commit.

Confirma la despreparación:

```
izanr@Izan MINGW64 ~/Desktop/Git (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:   documento.txt

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

Escenario C: Deshacer la última confirmación (commit)

Introduce un nuevo cambio y confírmalo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "Commit con error accidental" >> README.txt
git add README.txt
git commit -m "Commit con error accidental"
warning: in the working copy of 'README.txt', LF will be replaced by CRLF the ne
xt time Git touches it
[master fc17067] Commit con error accidental
 1 file changed, 1 insertion(+)
```

Verifica el historial:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git log --oneline
fc17067 (HEAD -> master) Commit con error accidental
2d6fa10 Añadida nueva linea al README y creado documento.txt
6dd37ba Primera versión del README
```

Deshaz la última confirmación (manteniendo los cambios):

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git reset --soft HEAD~1
```

El comando git reset --soft HEAD~1 mueve el puntero del historial al commit anterior pero mantiene los cambios en el área de preparación. Así puedo rehacer el último commit sin perder nada. Es útil cuando hago un commit con un mensaje incorrecto o antes de tiempo, ya que me permite corregirlo antes de confirmar de nuevo.

Verifica el estado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.txt
```

Añade más cambios y vuelve a confirmar correctamente:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "Contenido final del documento" >> documento.txt
git add documento.txt
git commit -m "Commit final y corregido de README y documento"
warning: in the working copy of 'documento.txt', LF will be replaced by CRLF the
next time Git touches it
[master 40cdb7c] Commit final y corregido de README y documento
  2 files changed, 3 insertions(+)
```

Verifica el historial final:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git log --oneline
40cdb7c (HEAD -> master) Commit final y corregido de README y documento
2d6fa10 Añadida nueva línea al README y creado documento.txt
6dd37ba Primera versión del README
```

Escenario D: Recuperar un archivo eliminado por error

Elimina un archivo del directorio de trabajo:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ rm documento.txt
```

Verifica el estado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
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:    documento.txt

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

Restaura el archivo eliminado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git restore documento.txt
```

Git puede recuperar archivos borrados porque guarda una copia completa en cada confirmación dentro del repositorio. Con git restore, se trae de vuelta la última versión confirmada del archivo, incluso si lo he eliminado físicamente del disco. Esto demuestra que Git actúa como una “copia de seguridad” de cada versión del proyecto.

Confirma la restauración:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Escenario E: Ignorar archivos (evitar "cagadas" futuras)

Crea un archivo temporal:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ echo "texto" > log.txt
```

Verifica su estado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    log.txt

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

Crea una regla de ignorado:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ printf "log.txt\n*.log\n" > .gitignore
```

Confirma que el archivo está ignorado:

```
izanr@Izan MINGW64 ~/Desktop/Git (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)
```

El archivo .gitignore sirve para evitar que Git rastree archivos temporales o generados automáticamente (por ejemplo, logs o compilaciones). Es importante porque mantiene el repositorio limpio y evita subir archivos innecesarios o personales. Una vez que un archivo está en .gitignore, ya no aparece como “no rastreado” en git status.

Confirma el archivo de ignorados:

```
izanr@Izan MINGW64 ~/Desktop/Git (master)
$ git add .gitignore
git commit -m "Añadido .gitignore para ignorar archivos temporales"
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it
[master b117269] Añadido .gitignore para ignorar archivos temporales
 1 file changed, 2 insertions(+)
 create mode 100644 .gitignore
```