## Revisando el Estado de tus Archivos

Supongamos que añades un nuevo archivo a tu proyecto, un simple README. Si el archivo no existía antes y ejecutas `git status`, verás el archivo sin rastrear de la siguiente manera:

```
$ echo 'My Project' > README

$ git status

On branch master

Untracked files:

  (use "git add <file>..." to include in what will be committed)



    README



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

Puedes ver que el archivo README está sin rastrear porque aparece debajo del encabezado “Untracked files” (“Archivos no rastreados” en inglés) en la salida. Sin rastrear significa que Git ve archivos que no tenías en el **`commit`** anterior. Git no los incluirá en tu próximo **`commit`** a menos que se lo indiques explícitamente. Se comporta así para evitar incluir accidentalmente archivos binarios o cualquier otro archivo que no quieras incluir. Como tú sí quieres incluir README, debes comenzar a rastrearlo.

## Rastrear Archivos Nuevos

Para comenzar a rastrear un archivo debes usar el comando `git add`. Para comenzar a rastrear el archivo README, puedes ejecutar lo siguiente:

```
$ git add README
```

Ahora si vuelves a ver el estado del proyecto, verás que el archivo README está siendo rastreado y está preparado para ser confirmado:

```
$ git status

On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)
```

```
    new file:   README
```

Puedes ver que está siendo rastreado porque aparece luego del encabezado “Cambios a ser confirmados” (“Changes to be committed” en inglés). Si confirmas en este punto, se guardará en el historial la versión del archivo correspondiente al instante en que ejecutaste `git add`.

## Preparar Archivos Modificados

Vamos a cambiar un archivo que esté rastreado. Si cambias el archivo rastreado llamado “CONTRIBUTING.md” y luego ejecutas el comando `git status`, verás algo parecido a esto:

```
$ git status

On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)



    new file:   README



Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in working
directory)



    modified:   CONTRIBUTING.md
```

El archivo “CONTRIBUTING.md” aparece en una sección llamada “Changes not staged for commit” (“Cambios no preparado para confirmar” en inglés) - lo que significa que existe un archivo rastreado que ha sido modificado en el directorio de trabajo pero que aún no está preparado. Para prepararlo, ejecutas el comando `git add`. `git add` es un comando que cumple varios propósitos - lo usas para empezar a rastrear archivos nuevos, preparar archivos, y hacer otras cosas como marcar archivos en conflicto por combinación como resueltos. Es más útil que lo veas como un comando para “añadir este contenido a la próxima confirmación” más que para “añadir este archivo al proyecto”. Ejecutemos `git add` para preparar el archivo “CONTRIBUTING.md” y luego ejecutemos `git status`:

```
$ git add CONTRIBUTING.md

$ git status
```

```
On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)



    new file:   README

    modified:   CONTRIBUTING.md
```

Ambos archivos están preparados y formarán parte de tu próxima confirmación. En este momento, supongamos que recuerdas que debes hacer un pequeño cambio en `CONTRIBUTING.md` antes de confirmarlo. Abres de nuevo el archivo, lo cambias y ahora estás listos para confirmar. Sin embargo, ejecutemos `git status` una vez más:

```
$ vim CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)



    new file:   README

    modified:   CONTRIBUTING.md



Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in working
directory)



    modified:   CONTRIBUTING.md
```

¡¿Pero qué…?! Ahora `CONTRIBUTING.md` aparece como preparado y como no preparado. ¿Cómo es posible? Resulta que Git prepara un archivo de acuerdo al estado que tenía cuando ejecutas el comando `git add`. Si confirmas ahora, se confirmará la versión de `CONTRIBUTING.md` que tenías la última vez que ejecutaste `git add` y no la versión que ves ahora en tu directorio de trabajo al ejecutar `git status`. Si modificas un archivo luego de ejecutar `git add`, deberás ejecutar `git add` de nuevo para preparar la última versión del archivo:

```
$ git add CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)



    new file:   README

    modified:   CONTRIBUTING.md
```

### Ver los Cambios Preparados y No Preparados

Supongamos que editas y preparas el archivo `README` de nuevo y luego editas `CONTRIBUTING.md` pero no lo preparas. Si ejecutas el comando `git status`, verás algo como esto:

```
$ git status

On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)



    new file:   README



Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in working
directory)



    modified:   CONTRIBUTING.md
```

Para ver qué has cambiado pero aun no has preparado, escribe `git diff` sin más parámetros:

```
$ git diff
```

```
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md

index 8ebb991..643e24f 100644

--- a/CONTRIBUTING.md

+++ b/CONTRIBUTING.md

@@ -65,7 +65,8 @@ branch directly, things can get messy.

 Please include a nice description of your changes when you submit your
PR;

 if we have to read the whole diff to figure out why you're contributing

 in the first place, you're less likely to get feedback and have your
change

-merged in.

+merged in. Also, split your changes into comprehensive chunks if you
patch is

+longer than a dozen lines.



 If you are starting to work on a particular area, feel free to submit a
PR

 that highlights your work in progress (and note in the PR title that it's
```

Este comando compara lo que tienes en tu directorio de trabajo con lo que está en el área de preparación. El resultado te indica los cambios que has hecho pero que aun no has preparado.

Si quieres ver lo que has preparado y será incluido en la próxima confirmación, puedes usar `git diff --staged`. Este comando compara tus cambios preparados con la última instantánea confirmada.

```
$ git diff --staged

diff --git a/README b/README

new file mode 100644

index 0000000..03902a1

--- /dev/null

+++ b/README

@@ -0,0 +1 @@
```

```
+My Project
```

Es importante resaltar que al llamar a `git diff` sin parámetros no verás los cambios desde tu última confirmación - solo verás los cambios que aun no están preparados. Esto puede ser confuso porque si preparas todos tus cambios, `git diff` no te devolverá ninguna salida.
Pasemos a otro ejemplo, si preparas el archivo `CONTRIBUTING.md` y luego lo editas, puedes usar `git diff` para ver los cambios en el archivo que ya están preparados y los cambios que no lo están. Si nuestro ambiente es como este:

```
$ git add CONTRIBUTING.md

$ echo 'test line' >> CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)



    modified:   CONTRIBUTING.md



Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in working
directory)



    modified:   CONTRIBUTING.md
```

Puedes usar `git diff` para ver qué está sin preparar

```
$ git diff

diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md

index 643e24f..87f08c8 100644

--- a/CONTRIBUTING.md

+++ b/CONTRIBUTING.md

@@ -119,3 +119,4 @@ at the

 ## Starter Projects
```

```
 See our [projects
list](https://github.com/libgit2/libgit2/blob/development/PROJECTS.md).

+# test line
```

y `git diff --cached` para ver que has preparado hasta ahora (--staged y --cached son sinónimos):

```
$ git diff --cached

diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md

index 8ebb991..643e24f 100644

--- a/CONTRIBUTING.md

+++ b/CONTRIBUTING.md

@@ -65,7 +65,8 @@ branch directly, things can get messy.

 Please include a nice description of your changes when you submit your
PR;

 if we have to read the whole diff to figure out why you're contributing

 in the first place, you're less likely to get feedback and have your
change

-merged in.

+merged in. Also, split your changes into comprehensive chunks if you
patch is

+longer than a dozen lines.



 If you are starting to work on a particular area, feel free to submit a
PR

 that highlights your work in progress (and note in the PR title that it's
```

### Confirmar tus Cambios

Ahora que tu área de preparación está como quieres, puedes confirmar tus cambios. Recuerda que cualquier cosa que no esté preparada - cualquier archivo que hayas creado o modificado y que no hayas agregado con `git add` desde su edición - no será confirmado. Se mantendrán como archivos modificados en tu disco. En este caso, digamos que la última vez que ejecutaste `git status` verificaste que todo estaba preparado y

que estás listo para confirmar tus cambios. La forma más sencilla de confirmar es escribiendo `git commit`:

```
$ git commit
```

Al hacerlo, arrancará el editor de tu preferencia. (El editor se establece a través de la variable de ambiente `$EDITOR` de tu terminal - usualmente es vim o emacs, aunque puedes configurarlo con el editor que quieras usando el comando `git config --global core.editor` tal como viste en [ch01-introduction]).

El editor mostrará el siguiente texto (este ejemplo corresponde a una pantalla de Vim):

```
# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# Changes to be committed:

#       new file:   README

#       modified:   CONTRIBUTING.md

#

~

~

~

".git/COMMIT_EDITMSG" 9L, 283C
```

Puedes ver que el mensaje de confirmación por defecto contiene la última salida del comando `git status` comentada y una línea vacía encima de ella. Puedes eliminar estos comentarios y escribir tu mensaje de confirmación, o puedes dejarlos allí para ayudarte a recordar qué estás confirmando. (Para obtener una forma más explícita de recordar qué has modificado, puedes pasar la opción `-v` a `git commit`. Al hacerlo se incluirá en el editor el diff de tus cambios para que veas exactamente qué cambios estás confirmando). Cuando sales del editor, Git crea tu confirmación con tu mensaje (eliminando el texto comentado y el diff).

Otra alternativa es escribir el mensaje de confirmación directamente en el comando `commit` utilizando la opción -m:

```
$ git commit -m "Story 182: Fix benchmarks for speed"

[master 463dc4f] Story 182: Fix benchmarks for speed

 2 files changed, 2 insertions(+)

 create mode 100644 README
```

¡Has creado tu primera confirmación (o **`commit`**)! Puedes ver que la confirmación te devuelve una salida descriptiva: indica cuál rama has confirmado (`master`), que **`checksum`** SHA-1 tiene el **`commit`** (`463dc4f`), cuántos archivos han cambiado y estadísticas sobre las líneas añadidas y eliminadas en el **`commit`**.

Recuerda que la confirmación guarda una instantánea de tu área de preparación. Todo lo que no hayas preparado sigue allí modificado; puedes hacer una nueva confirmación para añadirlo a tu historial. Cada vez que realizas un **`commit`**, guardas una instantánea de tu proyecto la cual puedes usar para comparar o volver a ella luego.