

# Bootcamp Java Developer

Fase 1 - Java Analyst  
Módulo 1

# Cambios en repositorios

# Inicializar un repositorio

Usualmente, se puede **obtener un repositorio de Git de dos maneras:**

- Convertir un directorio local que, actualmente, no esté bajo ningún control de versión a un repositorio de Git.
- Clonar un repositorio de Git existente desde algún otro lugar.

De cualquier forma, conseguiremos tener un repositorio de Git, en nuestra máquina local, listo para trabajar.



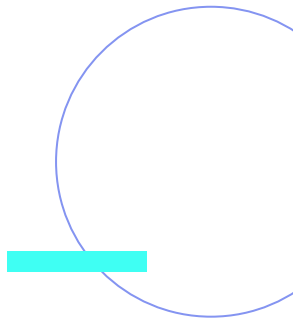
## git init

En el caso de tener un directorio de proyecto, que actualmente no está bajo ningún control de versión, y se desea comenzar a controlarlo con Git:

1. Primero, debemos **dirigirnos a ese directorio**.
2. Una vez allí, se debe correr el comando:

```
> git init
```

3. Esto va a **crear un nuevo subdirectorio** llamado **.git** que contiene todos los archivos necesarios para el repositorio - el esqueleto de un repositorio de Git-.



## git status

En este punto, nada en el proyecto está siendo controlado aún.

1. Si nuestro directorio está vacío porque acabamos de crearlo, vamos a crear un nuevo archivo dentro de él llamado ***info.txt***.
2. Ahora podemos ejecutar el siguiente comando:

```
> git status
```



### 3. Veremos el archivo sin seguimiento así:

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    info.txt

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

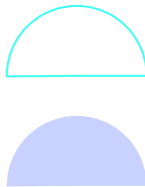
El archivo ***info.txt*** no tiene seguimiento ya que se encuentra debajo del titular “Untracked files”. ***Untracked***, básicamente, significa que Git ve un archivo que no tenía en la versión anterior del proyecto (*snapshot*). **Git no lo va a incluir hasta que no se lo indiquemos explícitamente.**

## git add

Para incluir y darle seguimiento a un archivo se puede usar el siguiente comando:

```
> git add info.txt
```

Si volvemos a ejecutar el comando de status nuevamente, veremos que nuestro archivo está bajo seguimiento y listo para ser confirmado, es decir, formar parte de la nueva versión del **repositorio(commit)**.



## Ejemplo

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   info.txt
```

Se advierte que está listo porque se encuentra debajo del titular **“Changes to be committed”**. Si generamos un commit en este punto, la versión del archivo en el momento en que hayamos ejecutado el comando **git add** va a permanecer en el snapshot histórico anterior.

El comando **git add** puede aceptar el nombre de ruta de un archivo o directorio; si es un directorio, el comando agrega todos los archivos en ese directorio recursivamente.



Si se modifica el archivo **info.txt**, que ya tenía seguimiento anteriormente, y se vuelve a ejecutar el comando **git status**, se observará algo similar a lo siguiente:

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   info.txt

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:   info.txt
```



El archivo ahora aparece también bajo la sección llamada “**Changes not staged for commit**”.

Significa que un archivo con seguimiento ha sido modificado en el directorio de trabajo (**working directory**) pero no ha sido preparado aún.

Podemos preparar un archivo para el próximo **commit** utilizando también el comando **git add**.

Podemos entonces percibir que este comando es multipropósito ya que nos permite darle seguimiento a archivos nuevos y además preparar archivos que ya tenían seguimiento para el próximo **commit**.

Se debe tener en cuenta que al momento de realizar un **commit**, la versión del archivo que formará parte va a ser la que estaba cuando se ejecutó el comando **git add**. Es decir que si se modifica un archivo luego de haber ejecutado el comando **git add**, se deberá ejecutar nuevamente el comando **git add** para agregar los cambios correspondientes.



Como alternativa podemos usar una versión interactiva del comando **add**:

```
> git add -i
```

Esta otra forma **brindará mucho más detalle y control sobre los cambios que podemos realizar** en un momento dado en el repositorio.

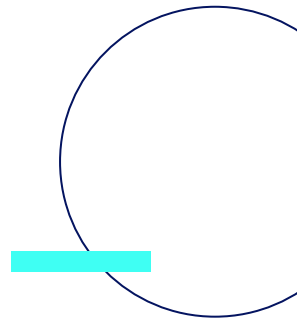
Para más información sobre cómo poder utilizar este comando, visitar: [git-scm.com](https://git-scm.com).



## git commit

Ahora que el archivo se encuentra en el área de preparación (**stage area**), estamos listos para **confirmar los cambios**, es decir: realizar un **commit**. La manera más corta es ejecutar el siguiente comando:

```
> git commit
```



Hacer esto lanzará el editor elegido, que mostrará lo siguiente:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#   new file:   info.txt
#   modified:   info.txt
#
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

Se observa que el mensaje por defecto del **commit** contiene la última salida del comando **git status** comentada y una línea vacía arriba de todo. Se pueden remover estos comentarios y escribir el mensaje o dejarlos como recordatorio de lo que se está confirmando en el **commit**.

Cuando salimos del editor, Git crea el **commit** con el mensaje.



Como alternativa, se puede escribir el mensaje del **commit** en la misma línea en la que se está ejecutando el comando mediante la especificación de la opción **-m** de la siguiente forma:

```
> git commit -m "Agrego cambios en el archivo"
[master 463dc4f] Agrego cambios en el archivo
1 file changed, 2 insertions(+)
create mode 100644 info.txt
```

Podemos observar que el comando nos ha dado información sobre sí mismo: la rama (**branch**) donde confirmamos los cambios (**master**), el identificador **SHA-1** del **commit** (**463dc4f**), cuántos archivos fueron modificados, y estadísticas acerca de las líneas que fueron insertadas o removidas en el **commit**.

Cada vez que hagamos un **commit** estaremos generando una nueva versión del proyecto, o **snapshot**, la cual podemos revertir o comparar luego.



Aunque el área de preparación puede ser útil para elaborar **commits** exactamente de la manera en que los queremos, muchas veces la misma puede ser muy compleja dado nuestro flujo de trabajo.

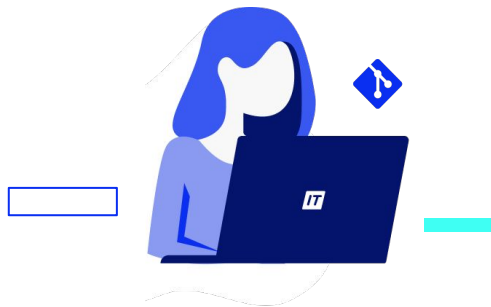
**Si queremos omitir el área de preparación**, Git viene con un atajo el cual nos permite automáticamente agregar todos los cambios realizados al área de preparación antes de realizar el **commit**, de esta manera podemos omitir el comando **git add** de la siguiente forma:

```
> git commit -a -m 'agrego nuevo cambio'
[master 83e38c7] agrego nuevo cambio
1 file changed, 5 insertions(+), 0 deletions(-)
```

## git commit -amend

Si lo que necesitamos es **editar el mensaje del último commit** que hayamos hecho porque hemos tenido un error, podemos hacerlo con el comando:

```
> git commit -amend
```




Este comando va a usar el mensaje del último commit que hayamos hecho y lo va a cargar en una sesión de edición en el editor que tengamos configurado en Git en donde podemos hacer los cambios que queramos, guardarlos y salir.

Cuando salgamos del editor, se va a escribir un nuevo commit el cual contiene nuestros cambios realizados.

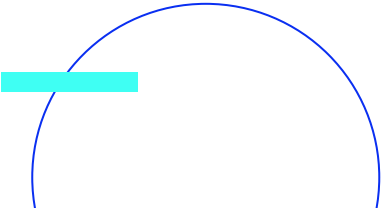




Tengamos en cuenta que si ya estamos trabajando con repositorios remotos (lo veremos más adelante en el curso) y hemos subido el último **commit** a nuestro servidor distribuido, no debemos cambiar nada del último **commit**, ya que al hacerlo estamos reemplazándolo con uno nuevo, el cual va a diferir del historial de **commits** que aparece en nuestro repositorio distribuido en el servidor, por lo cual los demás integrantes del proyecto que estén participando van a tener problemas en actualizarse.

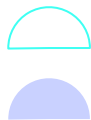


Para cambios más drásticos en nuestros **commit**, como por ejemplo cambiar el mensaje de **commits** anteriores al último, reordenamiento, borrado o integración de los mismos, necesitamos **herramientas de depuración de historial**, las cuales vamos a ver más adelante en el curso.



## git log

Luego de haber creado varios **commits**, o mismo si hemos clonado un repositorio con un historial de **commits** existente, probablemente queramos mirar hacia atrás para ver qué ha pasado en el repositorio. La herramienta más básica y poderosa para hacer esto es el comando **git log**.



```
> git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Horacio Gutierrez<mi.direccion.email@ejemplo.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
```

Cambio el número de versión

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Horacio Gutierrez<mi.direccion.email@ejemplo.com>
Date:   Sat Mar 15 16:40:33 2008 -0700
```

Remuevo test innecesario

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Horacio Gutierrez<mi.direccion.email@ejemplo.com>
Date:   Sat Mar 15 10:31:28 2008 -0700
```

Primer Commit

Por defecto, sin parámetros, **git log** lista los **commits** hechos en un repositorio en orden cronológico inverso; esto es, el **commit** más reciente se verá primero. Como podemos ver, este comando lista cada **commit** con su identificador SHA-1, el nombre de autor e email, la fecha de escritura y el mensaje del **commit**.

Este comando viene con muchos atajos y parámetros que podemos agregar para que la salida en la línea de comandos no sea tan abundante. Particularmente las opciones de **--oneline** y **--graph** son sumamente útiles para mostrar información abreviada sobre cada commit y para poder ver un gráfico ASCII mostrándonos nuestro historial de **commits**.



```
> git log --oneline --graph
* 2d3acf9 ignore errors from SIGCHLD on trap
* 5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
* | 5a09431 add timeout protection to grit
* | e1193f8 support for heads with slashes in them
|/
* d6016bc require time for xmlschema
* 11d191e Merge branch 'defunkt' into local
```

**¡Sigamos  
trabajando!**