



Aprendiendo desde cero

# TRABAJANDO CON GITHUB

Comandos del día a día



Adriana Arroyo Hernández



adrianapah10@gmail.com

# Descargando GIT

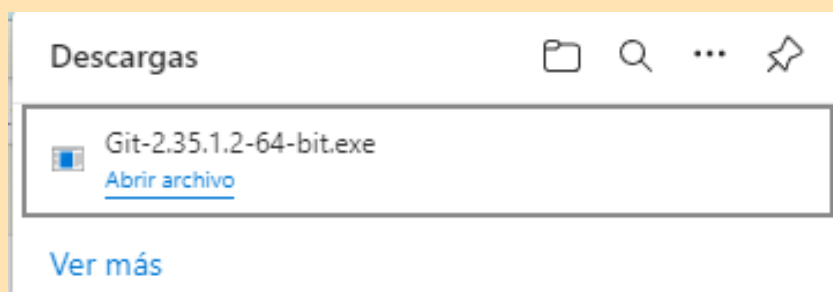
1. En el navegador colocamos "Download Git" o bien, vistamos <https://git-scm.com/downloads> y seleccionamos de acuerdo a nuestro sistema operativo.



2. Para windows seleccionamos la configuración de 64 bits.



3. Una vez descargado, ejecutamos el exe y damos siguiente..



# CONFIGURACIÓN DE USUARIO Y CORREO EN GIT



1. Buscamos gitBash o command prompt para conocer la versión del git instalado:

```
$ git --version  
git version 2.35.1.windows.2
```

2. Configuramos un nombre, este aparecerá en los commits:

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

3. Configuramos un correo:

```
git config --global user.email usuario@gmail.com
```

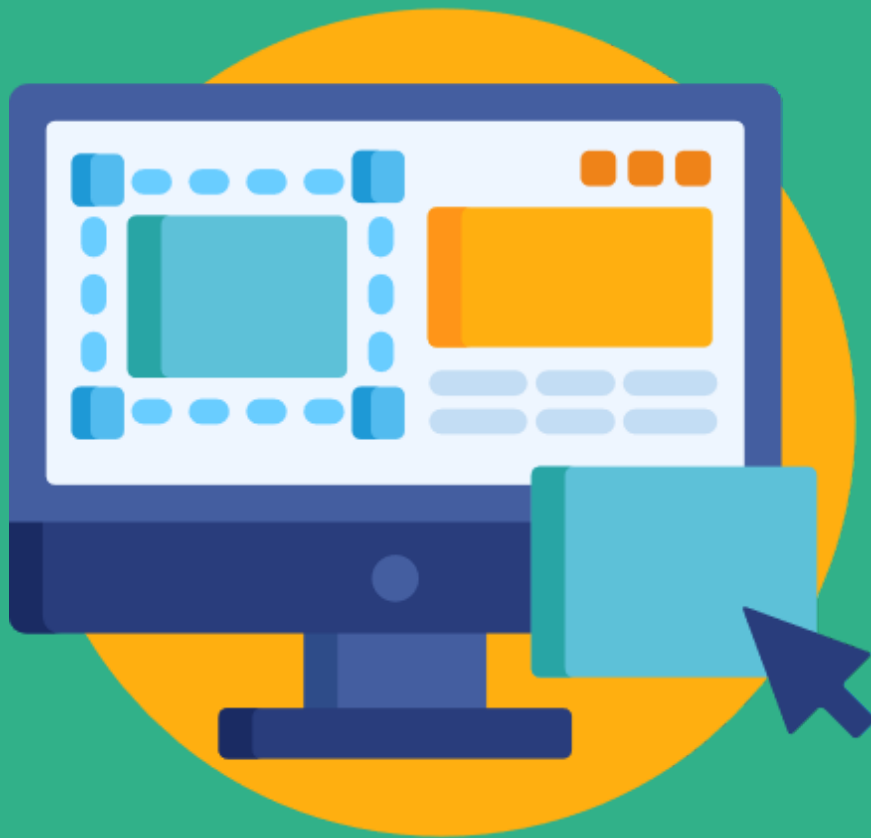
4. Comprobamos:

```
git config --list
```

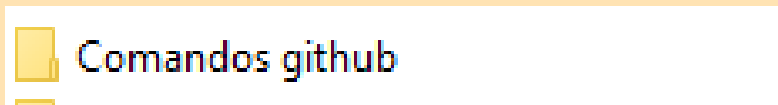
5. Para editar valores:

```
git config --global -e
```

# CREANDO REPOSITORIO LOCAL



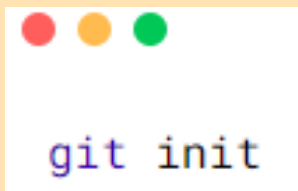
1. Escoger la carpeta que deseamos mantener vigilada:



2. Ingresar a la carpeta desde consola.:

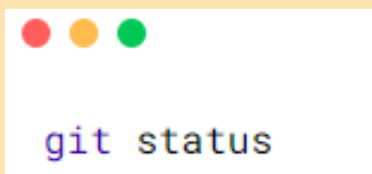
```
adria@DESKTOP-JN3E0VU MINGW64 ~  
$ cd '/d/Documentos/Comandos github'  
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Comandos github
```

3. Inicializar el repositorio:



```
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Comandos github  
$ git init  
Initialized empty Git repository in D:/Documentos/Comandos github/.git/
```

4. Comprobar archivos:

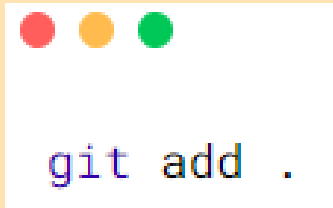


```
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Comandos github (main)  
$ git status  
On branch main  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    Estos son algunos comandos de git.txt
```

# AGREGAR ARCHIVOS AL REPOSITORIO LOCAL



1. Para agregar los nuevos archivos o actualizar las modificaciones ejecutamos:

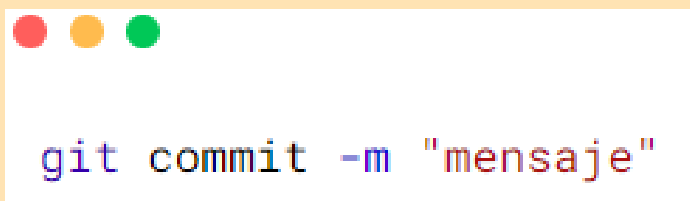


```
git add .
```

2. Ejecutamos nuevamente git status, y los archivos se mostrarán en verde:

```
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Estos son algunos comandos de git.txt
```

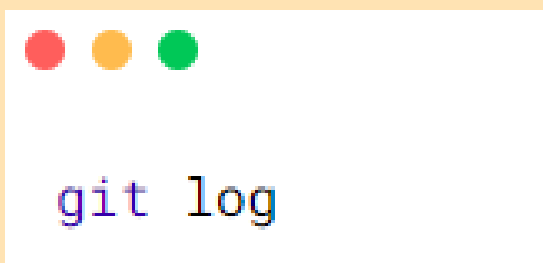
3. Colocamos un mensaje a los archivos agregados por si en un futuro queremos regresar a una versión anterior, saber a cual debemos regresar:



```
git commit -m "mensaje"
```

```
$ git commit -m "se agregó txt"
[main (root-commit) 6cd313c] se agregó txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Estos son algunos comandos de git.txt
```

4. Para ver todos los commits realizados:



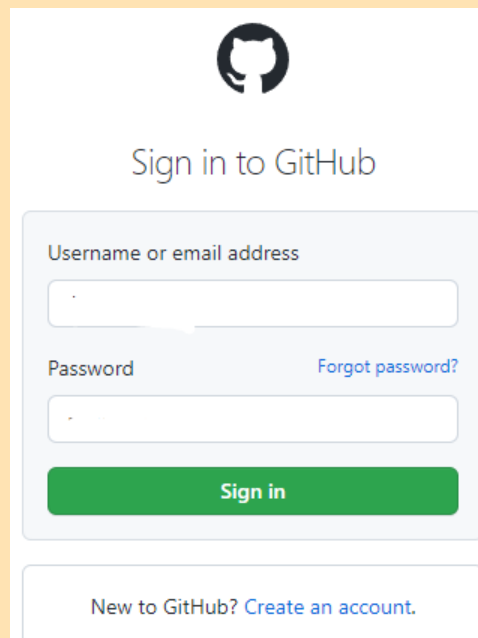
```
git log
```



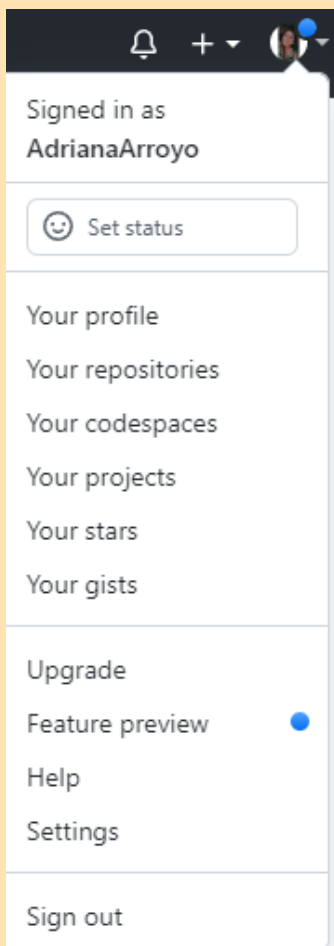
# AGREGAR ARCHIVOS AL REPOSITORIO EN LA NUBE



1. Ingresamos a <https://github.com>, si ya tenemos cuenta, solamente ingresamos, si no, deberemos crear una cuenta:



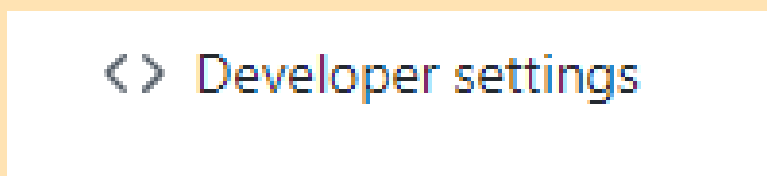
2. Una vez adentro de git, tocamos nuestra foto de perfil:



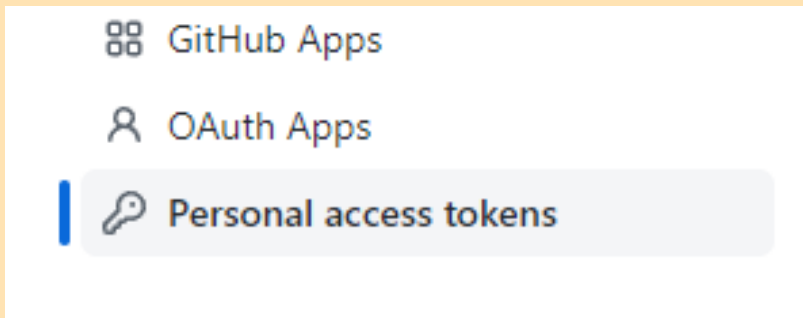
3. Seleccionamos settings:



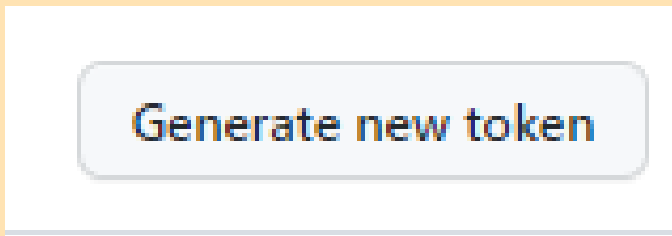
4. Seleccionamos Developer settings:



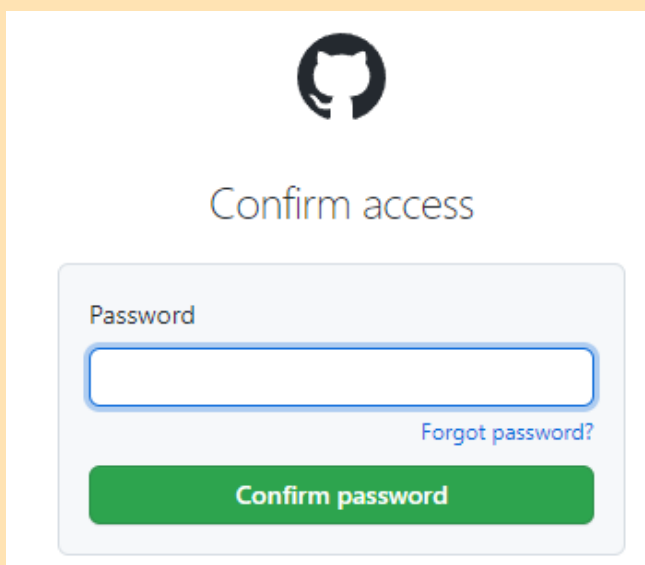
5. Seleccionamos Personal access tokens:



6. Seleccionamos generate new token:



7. Escribimos nuestra contraseña:



## 8. Establecemos un nombre para nuestro token, ajustamos expiración y creamos:

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

repositoriosRemotos

What's this token for?

Expiration \*

No expiration + The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure. [Learn more](#)

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> read:org	Read org and team membership, read org projects
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys

Generate token

9. Nos dará un token, el cual debemos copiar y guardar en algún lugar donde podamos recurrir a él, ya que desde github no se puede volver a ver cuál es.

Personal access tokens

Generate new token Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp\_CH9q4ZgT... Delete

Token para local — admin:enterprise, admin:pgp\_key, admin:org, admin:org\_hook, Last used within the last week Delete

admin:public\_key, admin:repo\_hook, delete:packages, delete\_repo, gist, notifications, repo, user, workflow, write:discussion, write:packages

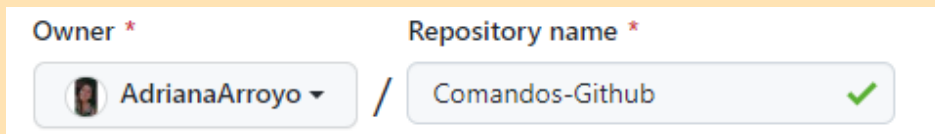
⚠ This token has no expiration date.

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

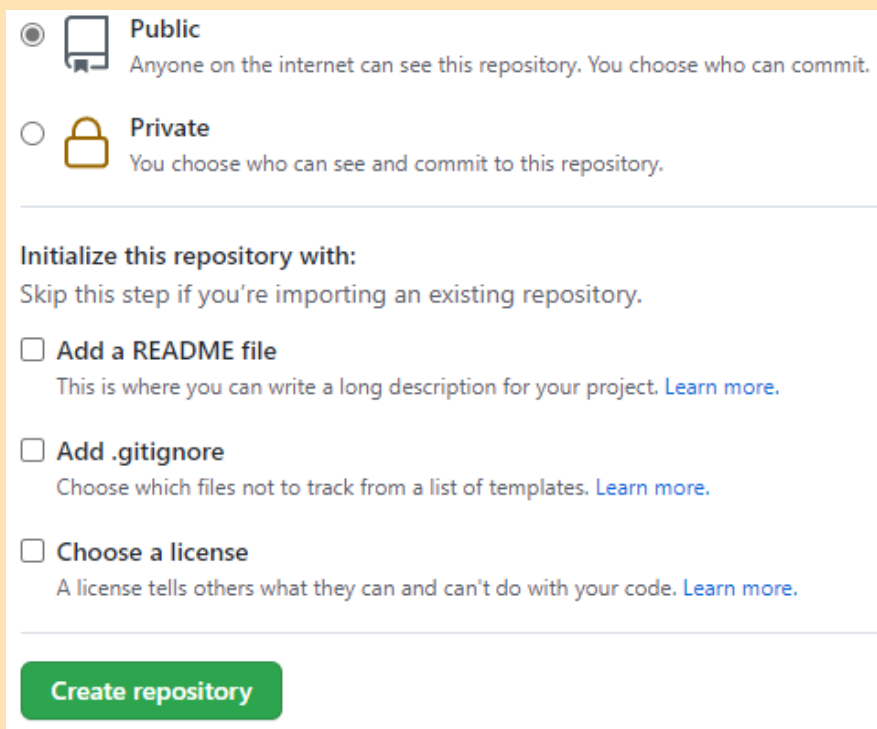
10. Una vez creado el token podemos continuar, creamos un nuevo repositorio:



11. Escribimos el nombre de nuestro repositorio:

A screenshot of a GitHub repository creation form. It has two main sections: "Owner" and "Repository name". The "Owner" section shows a dropdown menu with a profile picture and the name "AdrianaArroyo". The "Repository name" section shows a text input with "Comandos-Github" and a green checkmark to its right, indicating it is valid.

12. Escogemos la privacidad de nuestro proyecto, y creamos:

A screenshot of the GitHub repository creation form, showing the privacy and initialization options. The "Public" option is selected with a radio button. Below it, the "Private" option is shown with a radio button and a lock icon. Under the heading "Initialize this repository with:", there are three checkboxes: "Add a README file", "Add .gitignore", and "Choose a license". Each checkbox has a brief description and a "Learn more" link. At the bottom, there is a green "Create repository" button.

13. Al crearlo aparecerán los siguientes comandos:

#### ...or create a new repository on the command line

```
echo "# Comandos-Github" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/AdrianaArroyo/Comandos-Github.git
git push -u origin main
```

#### ...or push an existing repository from the command line

```
git remote add origin https://github.com/AdrianaArroyo/Comandos-Github.git
git branch -M main
git push -u origin main
```

#### ...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

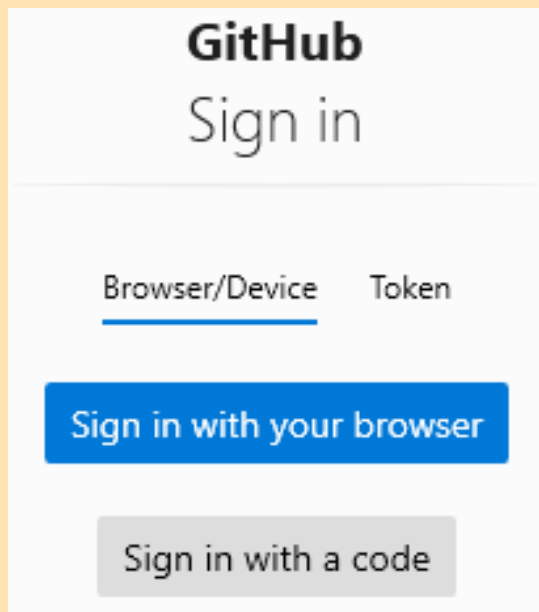
14. Tomamos las líneas de código que corresponden a un repositorio o folder existente y copiamos:

#### ...or push an existing repository from the command line

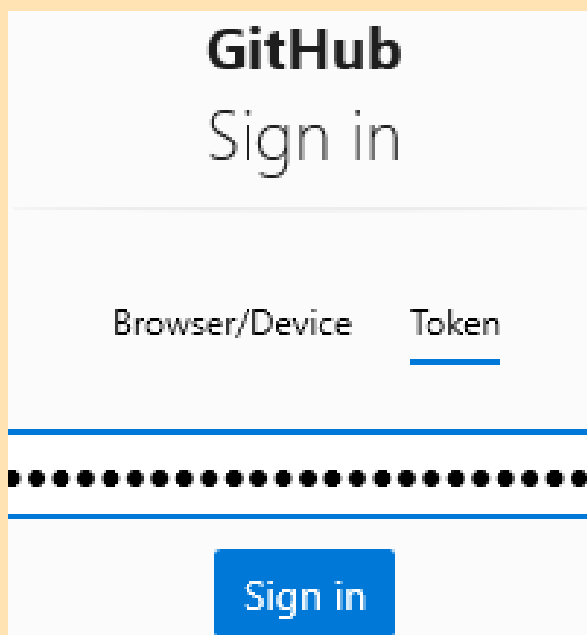
```
git remote add origin https://github.com/AdrianaArroyo/Comandos-Github.git
git branch -M main
git push -u origin main
```

El comando **git push** sube mis cambios al repositorio de la nube.

15. Pegamos en la consola y ENTER, nos pedirá autenticarnos:



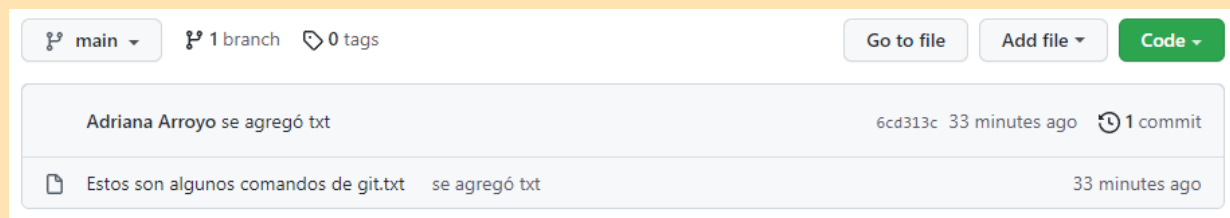
16. Ingresamos el token anteriormente creado y presionamos enter:



17. Una vez validado el token, se subirán los archivos a nuestro repositorio en la nube:

```
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Comandos github (main)
$ git remote add origin https://github.com/AdrianaArroyo/Comandos-Github.git
git branch -M main
git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 246 bytes | 123.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AdrianaArroyo/Comandos-Github.git
* [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

18. Refrescamos la página de github y aparecerán nuestros archivos:



19. Si deseamos agregar más archivos, debemos realizar el mismo proceso:

**git add .**

**git commit -m "mensaje"**

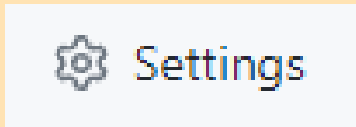
y para subir a la nube **git push**



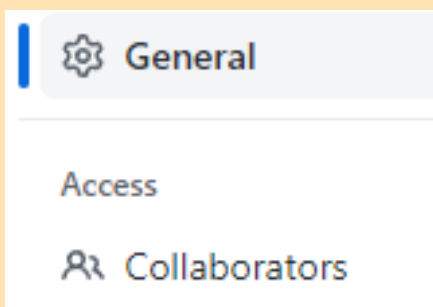
# INVITAR A UN MIEMBRO A COLABORAR



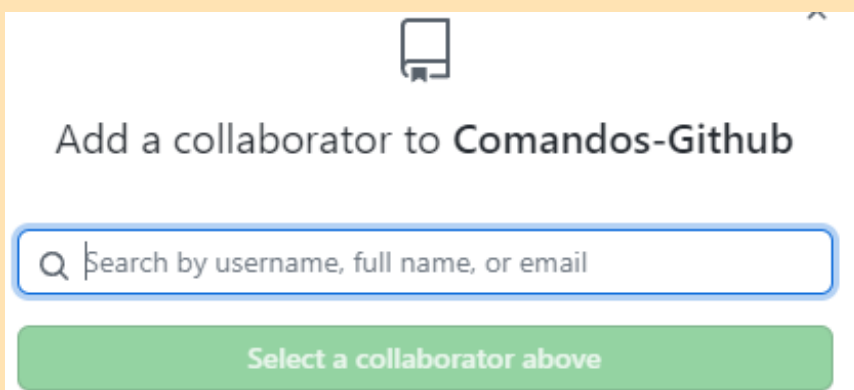
1. Buscamos en la pagina de nuestro repositorio, la siguiente opción:



2. Dentro de settings, buscaremos miembros o colaboradores:



3. Agregamos el correo del compañero a quien deseamos invitar:

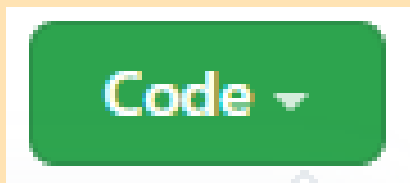


4. Para que pueda editar y subir código, es importante otorgarle el rol de "Mantainer"

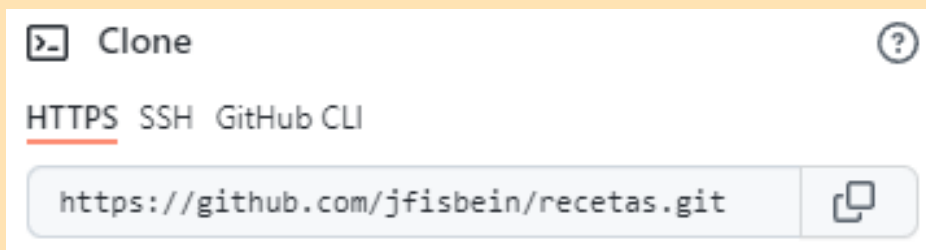
# CLONAR REPOSITÓRIO



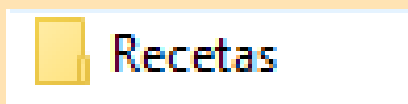
1. Buscamos el botón de code:



2. Copiamos la dirección https:

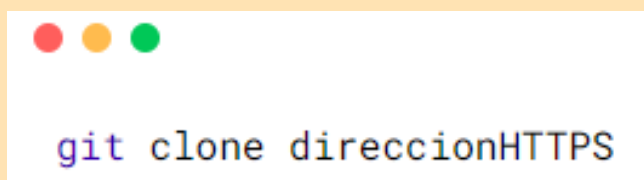


3. Escogemos una carpeta donde queremos guardar nuestro proyecto e ingresamos a ella por medio de la consola :



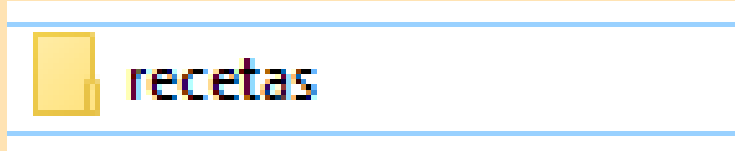
```
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Comandos github (main)
$ cd /d/Documentos/Recetas
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Recetas
```

4. Clonamos el proyecto:



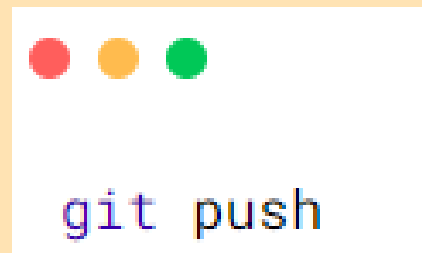
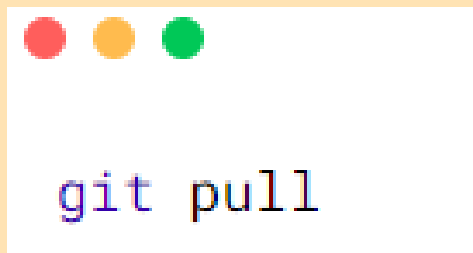
```
adria@DESKTOP-JN3E0VU MINGW64 /d/Documentos/Recetas
$ git clone https://github.com/jfisbein/recetas.git
Cloning into 'recetas'...
remote: Enumerating objects: 193, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 193 (delta 25), reused 50 (delta 25), pack-reused 143
Receiving objects: 100% (193/193), 30.86 KiB | 509.00 KiB/s, done.
Resolving deltas: 100% (108/108), done.
```

5. Buscamos dentro de nuestra carpeta, el proyecto clonado:

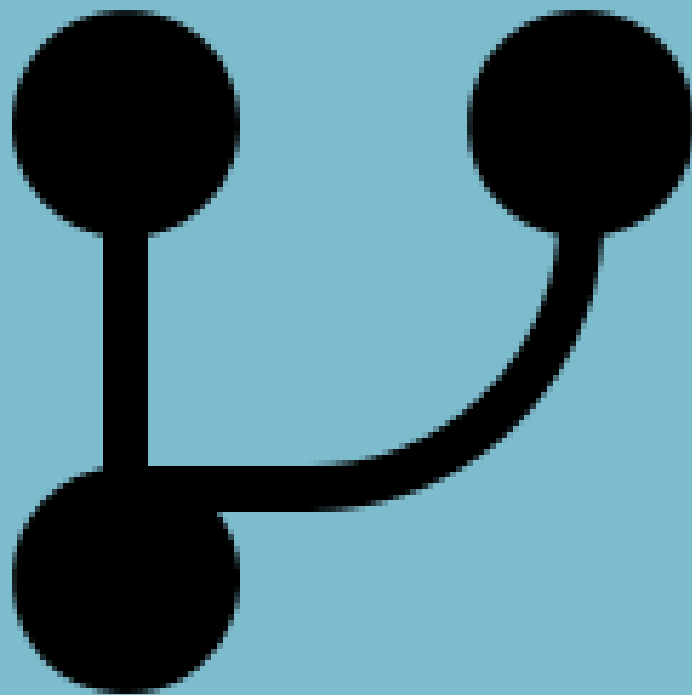


IMPORTANTE: Al clonar un proyecto en el que no somos colaboradores no podemos realizar cambios a menos que se realice un **FORK**.


Cada vez que vayamos a realizar un push, debemos primero realizar un pull, esto para actualizar nuestro repositorio en caso de que algún compañero haya agregado cambios al proyecto.



# RAMAS




Crear una rama:




```
git branch nombreRama
```

Ver ramas creadas:




```
git branch
```

Cambiar nombre de la rama:




```
git branch -m nomActual nomNuevo
```

Cambiar la rama en la que deseamos trabajar:




```
git checkout nomRama
```

Ver las diferencias de la rama actual y la principal:




```
git diff ramaActual ramaPrincipal
```

Para que el nombre de la rama principal siempre se configure por defecto a un nombre en específico:




```
git config --global init.defaultBranch nomDeseado
```

Eliminar rama de repositorio local:




```
git branch -d nombreRama
```

Subir rama al repositorio remoto:




```
git push --set-upstream origin nombreRama
```

Para observar todas las ramas creadas en el repositorio remoto:



```
git branch --all
```

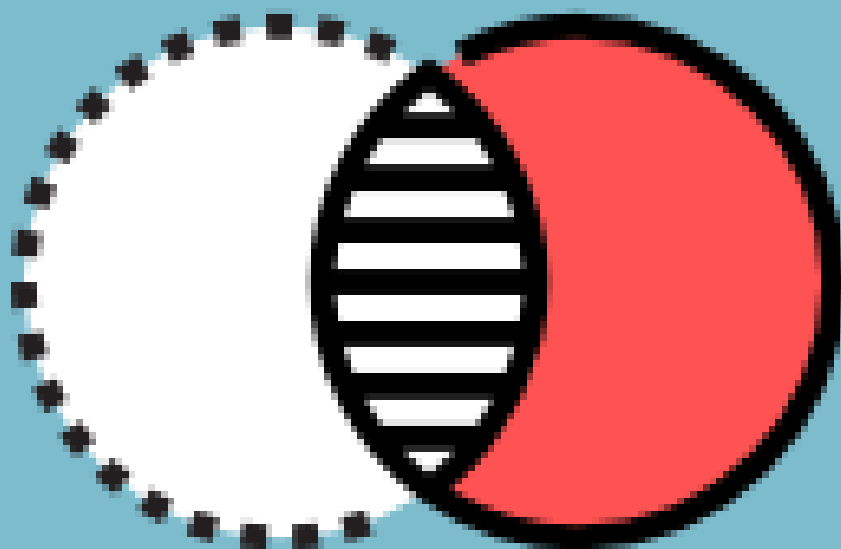
Traer a mi repositorio local todas las ramas creadas en el repositorio remoto:



```
git pull --all
```



# UNIONES




# Tipos de Uniones

- **Fast forward:** Es donde no se presentan cambios en la rama principal, por lo que puede unirse.
- **Unión automática:** Git detecta que en la rama principal hubo algún cambio que la otra rama desconoce, pero al no modificar líneas iguales, se puede realizar la unión.
- **Manual:** Git no puede resolver la unión de forma automática, ya que en ambas ramas se modificaron las mismas líneas, para ello deben resolverse las diferencias manualmente.


# FAST FORWARD

1. Regresamos a nuestra rama principal:




```
git checkout ramaPrincipal
```

2 . Realizamos la unión:



```
git merge ramaSecundaria
```

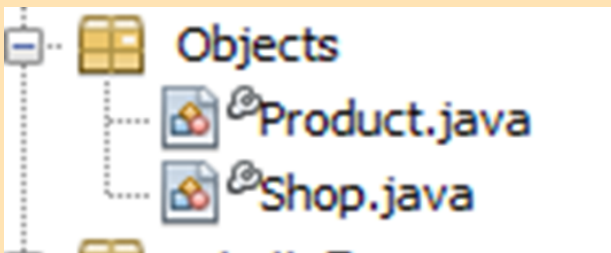
3. Si no necesitamos más la rama secundaria despues del merge, la podemos eliminar:



```
git branch -d ramaSecundaria
```

# UNIÓN AUTOMÁTICA

1. Supongamos que en la rama rama-producto vamos a trabajar en el Product pero en la master trabajaremos en shop:



2. Al realizar la unión ocurrirá lo siguiente:

```
Merge branch 'rama-producto'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
```

3. Para salir de aquí debemos apretar el botón de ESC :wq ENTER y debe aparecer el mensaje de unión :

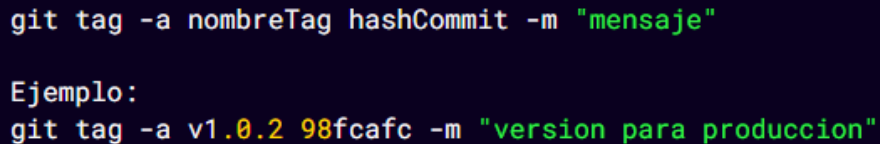
```
adria@DESKTOP-5JVPQS4 MINGW64 /d/Progra11/Lab1_B90755_Adriana_C06456_Jonathan (master)
$ git merge rama-producto
Merge made by the 'recursive' strategy.
 src/Objects/Product.java | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

# UNIÓN MANUAL

Si en ambas ramas se modificó el mismo archivo, deberemos resolver el conflicto de manera manual, para ello, abriremos el archivo en visual studio o un block de notas donde podremos comparar las diferencias y escoger la versión más adecuada, una vez el archivo se encuentre de la forma que deseamos, se procede a realizar un git add y git commit como normalmente se realiza.

# TAGS

Son la referencia a un commit específico en el tiempo, permitiendo marcar las versiones, para crear un tag, ejecutamos:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) at the top left. It contains the command `git tag -a nombreTag hashCommit -m "mensaje"` and an example: `Ejemplo: git tag -a v1.0.2 98fcafc -m "version para produccion"`.

```
git tag -a nombreTag hashCommit -m "mensaje"

Ejemplo:
git tag -a v1.0.2 98fcafc -m "version para produccion"
```

Ver los tags creados:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) at the top left. It contains the command `git tag`.

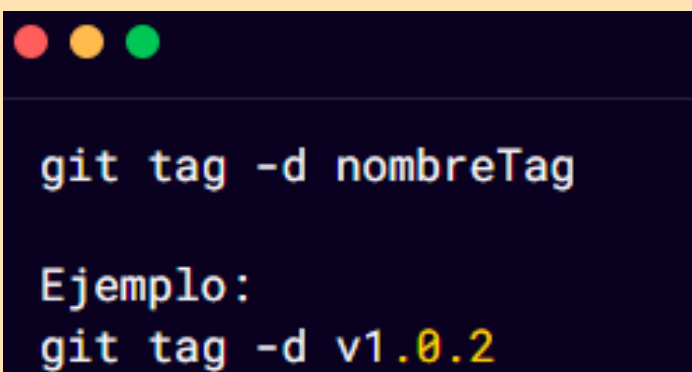
```
git tag
```

Ver un tag específico:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) at the top left. It contains the command `git show nombreTag`.

```
git show nombreTag
```

Eliminar un tag:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) at the top left. It contains the command `git tag -d nombreTag` and an example: `Ejemplo: git tag -d v1.0.2`.

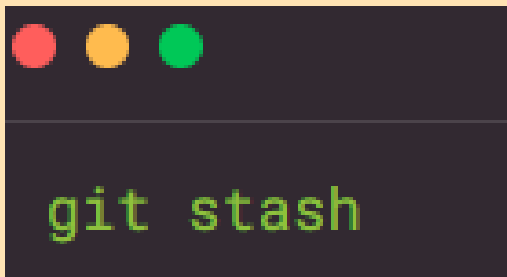
```
git tag -d nombreTag

Ejemplo:
git tag -d v1.0.2
```

# STASH

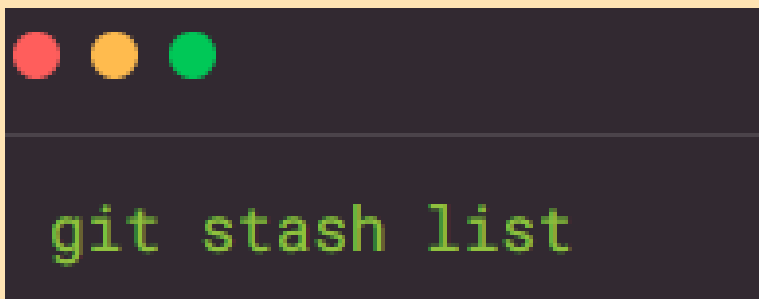
Es una “bóveda” donde se pueden guardar algunos cambios que aún no están listos para ser desplegados.

Crear stash:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text `git stash` is displayed in a green monospace font.

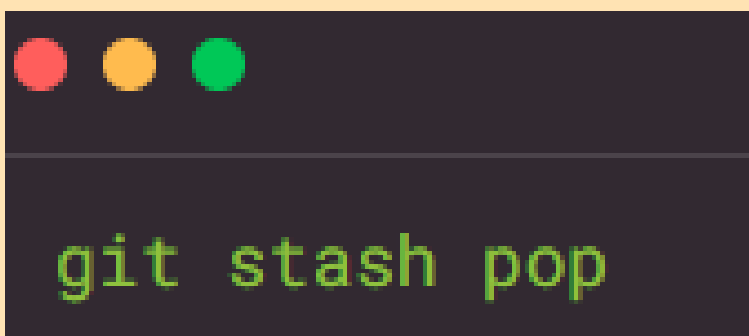
```
git stash
```

Ver los stash creados:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text `git stash list` is displayed in a green monospace font.

```
git stash list
```

Eliminar el stash y unirlo a la rama:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text `git stash pop` is displayed in a green monospace font.

```
git stash pop
```

# REBASE - ACTUALIZAR RAMA

Se usa cuando deseamos agregar commits de la rama principal, en la otra rama.

Crear rebase:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The command `git rebase ramaPrincipal` is displayed in a monospaced font, with 'git' in orange, 'rebase' in blue, and 'ramaPrincipal' in white.

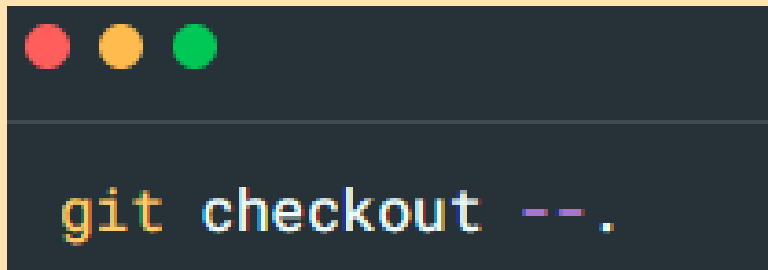
```
git rebase ramaPrincipal
```



# RECUPERACIÓN DE CAMBIOS EN REPOSITORIO

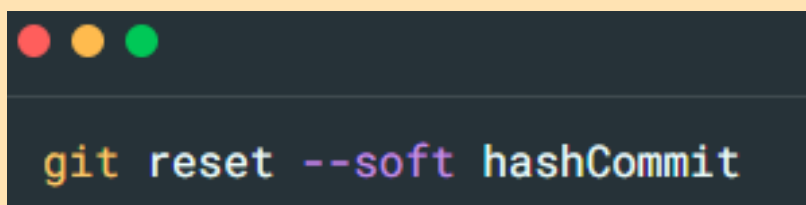


Regresar al último cambio que realizamos en nuestro proyecto sin eliminar código realizado:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The command `git checkout --.` is written in a monospaced font with syntax highlighting: `git` is orange, `checkout` is blue, `--` is purple, and `.` is white.

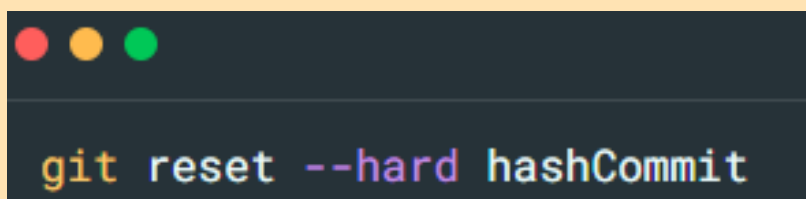
```
git checkout --.
```

Revertir commits posteriores al seleccionado:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The command `git reset --soft hashCommit` is written in a monospaced font with syntax highlighting: `git` is orange, `reset` is blue, `--soft` is purple, and `hashCommit` is white.

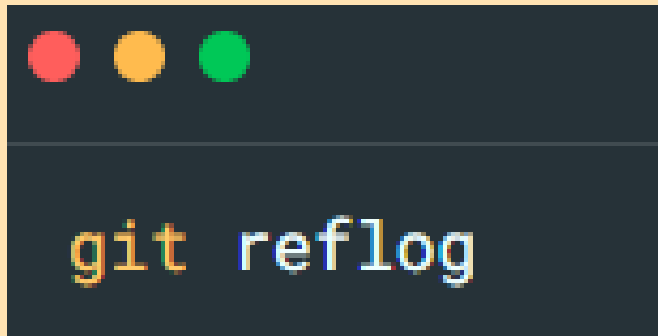
```
git reset --soft hashCommit
```

Revertir todos los cambios del código después de realizados los commits, y regresar a un punto deseado:

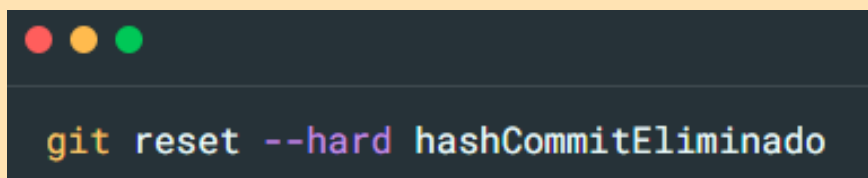
A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The command `git reset --hard hashCommit` is written in a monospaced font with syntax highlighting: `git` is orange, `reset` is blue, `--hard` is purple, and `hashCommit` is white.

```
git reset --hard hashCommit
```

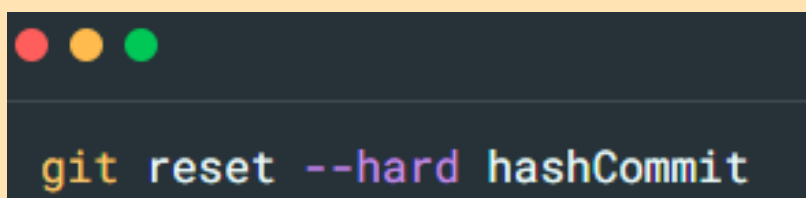
Observar en orden cronológico los cambios realizados:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'git reflog' is displayed in a monospaced font, with 'git' in orange and 'reflog' in light blue.

Una vez encontrado el commit eliminado que queremos recuperar, tomamos su hash y ejecutamos:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'git reset --hard hashCommitEliminado' is displayed in a monospaced font, with 'git' in orange, 'reset' in light blue, '--hard' in purple, and 'hashCommitEliminado' in light blue.

Revertir todos los cambios del código después de realizados los commits, y regresar a un punto deseado:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'git reset --hard hashCommit' is displayed in a monospaced font, with 'git' in orange, 'reset' in light blue, '--hard' in purple, and 'hashCommit' in light blue.


# ALGUNOS COMANDOS



# AGREGAR

Para subir todos los archivos usamos git add .

Sin embargo, si deseamos agregar solamente un archivo podemos hacerlo con:




```
git add nomArchivovl.extension
```

Por ejemplo:




```
git add README.md
```

Agregar todos los archivos de una misma extensión:



```
git add *.extension
```

Por ejemplo:



```
git add *.js
```

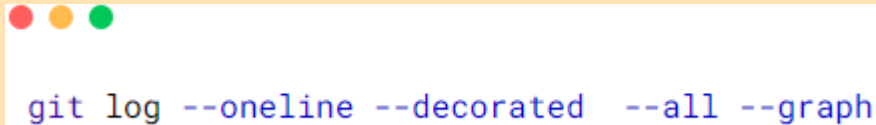
Ingresar toda una carpeta:



```
git add css/
```

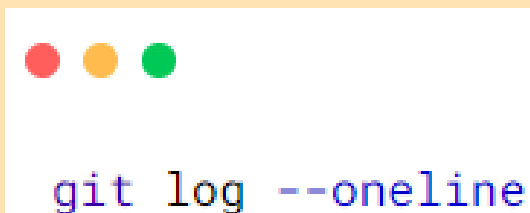
# LOGS

git log --oneline --decorated --all --graph



```
git log --oneline --decorated --all --graph
```

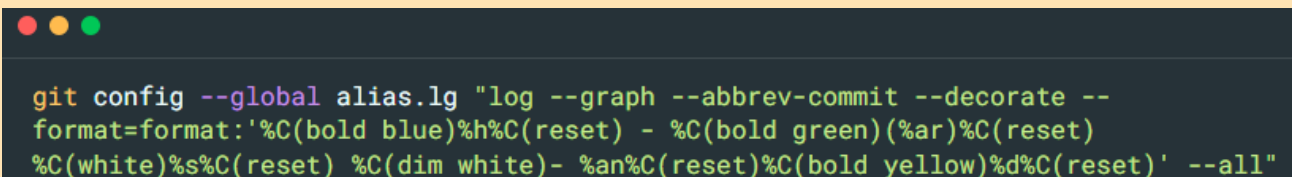
git log --oneline



```
git log --oneline
```

Crear un alias para un lg más bonito:

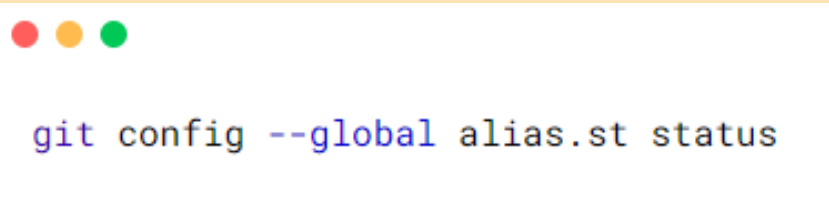
```
git config --global alias.lg "log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all"
```



```
git config --global alias.lg "log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all"
```

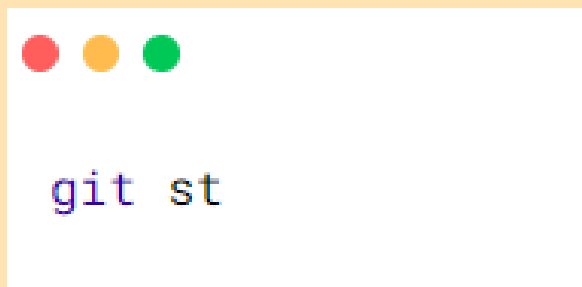
# OTROS

Los alias son utilizados para llamar a los comandos de git de una manera más corta, ejemplo:

A terminal window with a white background and three colored window control buttons (red, yellow, green) in the top-left corner. The text `git config --global alias.st status` is displayed in a monospaced font, with `git` in purple and the rest in blue.

```
git config --global alias.st status
```

Ahora cada vez que queramos comprobar el estado ejecutamos:

A terminal window with a white background and three colored window control buttons (red, yellow, green) in the top-left corner. The text `git st` is displayed in a monospaced font, with `git` in purple and `st` in blue.

```
git st
```

Verificar si en el repositorio de la nube ya se han elaborado cambios o aún no es necesario realizar un git pull:

A terminal window with a white background and three colored window control buttons (red, yellow, green) in the top-left corner. The text `git fetch` is displayed in a monospaced font, with `git` in purple and `fetch` in blue.

```
git fetch
```