

# Git

---

## Comandos

---

### Ayuda sobre un comando

```
# Abrir el navegador con información sobre un comando:  
$ git command --help
```

## Configuración

### Mostrar la configuración actual

```
git config --list
```

### Mostrar la configuración local

```
git config --local --list
```

### Mostrar la configuración global

```
git config --global --list
```

### Mostrar la configuración del sistema

```
git config --system --list
```

### Establecer un nombre de usuario identificable de forma global

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

### Establecer una dirección de correo identificable de forma global

```
git config --global user.email "your_mail@mail.com"
```

### Establecer coloreado automático de la línea de comandos de Git para una fácil revisión

```
git config --global color.ui auto
```

### Establecer el editor global para commits

```
git config --global core.editor vi
git config --global core.editor "code --wait"
```

## Establecer la rama por defecto

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

## Archivos de Configuración

### Archivo de configuración específico del repositorio [--local]

```
<repo>/ .git/config
```

### Archivo de configuración específico del usuario [--global]

```
~/.gitconfig
```

### Archivo de configuración del sistema [--system]

```
/etc/gitconfig
```

## Crear

### Clonar un repositorio existente

Existen dos maneras:

Vía SSH:

```
git clone ssh://usuario@dominio.com/repo.git
```

Vía HTTP:

```
git clone http://dominio.com/usuario/repo.git
```

### Crea un nuevo repositorio local

```
git init
```

### Realizar un *clone* de forma local desde la carpeta del proyecto a otra carpeta

```
git clone . /folder/to/the/project
```

---

## Cambios Locales

### Cambios en el directorio de trabajo

```
git status
```

### Cambios en archivos rastreados

```
git diff
```

### Agregar todos los cambios actuales al siguiente commit

```
git add .
```

### Agregar algunos cambios de <archivo> para el siguiente commit

```
git add -p <archivo>
```

### Realizar un commit de todos los cambios locales en los archivos rastreados

```
git commit -a
```

### Realizar un commit de los cambios previamente almacenados en el área de pruebas (stage area)

```
git commit
```

### Realizar un commit con un mensaje

```
git commit -m 'Aquí el mensaje'
```


### Realizar un commit saltándose el área de pruebas y agregando un mensaje

```
git commit -am 'aquí el mensaje'
```

### Realizar un commit a alguna fecha anterior

```
git commit --date=""date --date='n day ago'`" -am "Mensaje del commit"
```

### Cambiar último commit

 **NOTA:** ¡No modificar commits ya publicados!

```
git commit -a --amend
```

## Cambiar fecha del último commit

```
GIT_COMMITTER_DATE="date" git commit --amend
```

## Cambiar fecha del autor del último commit

```
git commit --amend --date="date"
```

## Mover cambios no confirmados (uncommitted changes) de la rama actual a otra rama

```
git stash
git checkout branch2
git stash pop
```

## Restaurar cambios del área de pruebas (stage area) a la rama actual

```
git stash apply
```

## Eliminar la última serie de cambios del área de pruebas (stage area)

```
git stash drop
```

## Buscar

### Un texto en todos los archivos del directorio

```
git grep "Hola"
```

### Un texto en cualquier versión

```
git grep "Hola" v2.5
```

## Historial de Commits

Mostrar todos los commits, empezando por los más recientes (se mostrará el hash, información sobre el autor, fecha y título del commit)

```
git log
```

## Mostrar todos los commits (sólo se mostrará el hash y el mensaje del commit)

```
git log --oneline
```

## Mostrar todos los commits de un usuario específico

```
git log --author="usuario"
```

## Mostrar los cambios a través del tiempo de un archivo específico

```
git log -p <archivo>
```

## Mostrar commits que están presentes sólomente en remote/branch al lado derecho

```
git log --oneline <origin/master>..  
<remote/master> --left-right
```

## Quién cambió, qué y cuándo en 'archivo'

```
git blame <archivo>
```

## Mostrar reference log

```
git reflog show
```

## Borrar reference log

```
git reflog delete
```

## Mostrar el SHA1 del último commit de cada ref

```
git ls-remote <repo>
```

## Ramas & Etiquetas

### Listar todas las ramas locales

```
git branch
```

### Listar todas las ramas remotas

```
git branch -r
```

### Listar todas las ramas locales y remotas

```
git branch -a
```

### Listar sólo las ramas remotas y el commit

```
git branch -rv
```

### Cambiar rama HEAD

```
git checkout <rama>
```

### Crear nueva rama y cambiar a esta

```
git checkout -b <rama>
```

### Crear nueva rama basada en la rama HEAD actual

```
git branch <nueva-rama>
```


### Crear nueva rama de seguimiento basada en una rama remota

```
git branch --track <nueva-rama> <repositorio>/<rama-remota>
```

### Eliminar una rama local

```
git branch -d <rama>
```

### Forzar eliminación de una rama local

 **NOTA:** ¡Se perderán los cambios sin fusionar!

```
git branch -D <branch>
```

### Borrar rama remota

```
git push [remote] :[branch]
```

## Marcar el commit actual con una etiqueta

```
git tag <tag-name>
```

## Marcar el commit actual con una etiqueta que incluye un mensaje

```
git tag -a <etiqueta>
```

## Borrar la etiqueta local '12345'

```
git tag -d 12345
```

## Borrar la etiqueta remota '12345' (eg, GitHub version too)

```
git push origin :refs/tags/12345
```

Otra forma:

```
git push --delete origin tagName
```

## Actualizar & Publicar

### Listar todos los repositorios remotos configurados actuales

```
git remote -v
```

### Mostrar información sobre un remoto

```
git remote show <remoto>
```

### Agregar un nuevo repositorio, nombrado 'remoto'

```
git remote add <remoto> <url>
```

### Descargar todos los cambios de <remoto>, pero no integrarlos al HEAD

```
git fetch <remoto>
```

### Descargar cambios y fusionarlos/integrarlos directamente al HEAD

```
git remote pull <remote> <url>
```

## Obtener todos los cambios del HEAD al repositorio local

```
git pull origin master
```

## Obtener todos los cambios del HEAD al repositorio local sin fusionar

```
git pull --rebase <remote> <branch>
```

## Publicar cambios locales en un remoto

```
git push remote <remoto> <rama>
```

## Eliminar una rama en el remoto

```
git push <remoto> :<rama> (desde Git v1.5.0)  
git push <remoto> --delete <rama> (desde Git v1.7.0)
```

## Publicar tus etiquetas

```
git push --tags
```

## Fusionar y *Rebase*

### Fusionar "rama" en tu HEAD actual

```
git merge <rama>
```

### Hacer un *rebase* de tu actual HEAD sobre "rama"

⚠ **NOTA:** ¡No hacer *rebase* de commits ya publicados!

```
git rebase <rama>
```

### Aborta un *rebase*

```
git rebase --abort
```

### Continuar un *rebase* después de resolver conflictos



```
git rebase --continue
```

## Usar tu herramienta de fusión configurada para resolver conflictos

```
git mergetool
```

## Usar tu editor para manualmente resolver conflictos y (después de resueltos) marcar el archivo como resuelto

```
git add <archivo-resuelto>
```

```
git rm <archivo-resuelto>
```

## Aplastando commits (squashing)

```
git rebase -i <commit-just-before-first>
```

Ahora reemplazando esto,

```
pick <commit_id>
pick <commit_id2>
pick <commit_id3>
```

con esto,

```
pick <commit_id>
squash <commit_id2>
squash <commit_id3>
```

## Deshacer

### Descartar todos los cambios locales en tu directorio de trabajo

```
git reset --hard HEAD
```

### Sacar todos los archivos del área de pruebas (es decir, deshacer el último `git add`)

```
git reset HEAD
```

### Descartar cambios locales de un archivo específico

```
git checkout HEAD <archivo>
```

## Desechar todos los cambios de los archivos

```
git checkout .
```

## Revertir un commit (produciendo un nuevo commit con los cambios contrarios)

```
git revert <commit>
```

## Reestablecer tu puntero HEAD a un commit anterior y descartar todos los cambios desde entonces

```
git reset --hard <commit>
```

## Volver al índice de un commit (HEAD@{index})

```
git reset HEAD@{index}
```

## Reestablecer tu puntero HEAD al estado actual de una rama remota

```
git reset --hard <remote/branch> es decir, upstream/master, origin/my-feature
```

## Reestablecer tu puntero HEAD a un commit anterior y preservar todos los cambios en el área de pruebas (stage area)

```
git reset <commit>
```

## Reestablecer tu puntero HEAD a un commit anterior y preservar los cambios locales sin confirmar (uncommitted changes)

```
git reset --keep <commit>
```

## Remover los archivos que fueron accidentalmente agregados al commit antes de ser añadidos al fichero '.gitignore'

```
git rm -r --cached .  
git add .  
git commit -m "remove xyz file"
```

## Extra

---

## Resumen de comandos para nuevo repositorio

```
git init
git add --all
git commit -m "First Commit"
git tag -a v1.0.0 -m "Message"
git remote add origin git@bitbucket.org:usuario/repositorio.git
git push -u origin master
git clone ssh://usuario@servidor:/folder/to/project/myProject.git
git push [remote] --all
git push [remote] --tags
```

## Actualizar un 'fork' de un proyecto remoto

Mantener el 'fork' de un proyecto actualizado con los cambios del autor original:

```
# Clonar nuestro fork del otro proyecto
$ git clone git@github.com:YOUR-USERNAME/YOUR-FORKED-REPO.git

# Añadir la URL del repositorio original
$ cd into/cloned/fork-repo
$ remote add upstream git://github.com/ORIGINAL-DEV-USERNAME/REPO-YOU-FORKED-FROM.git
$ git fetch upstream

# Actualizar su bifurcación desde el repositorio original para mantenerse al día con sus cambios
$ git pull upstream master
```

## Cambiar commits

Cambiar la dirección de correo de todos los commits de un repositorio:

```
git filter-branch -f --env-filter '
    if test "$GIT_AUTHOR_EMAIL" != "your_email@mail.com"
    then
        GIT_AUTHOR_EMAIL=your_email@mail.com
    fi
    if test "$GIT_COMMITTER_EMAIL" != "your_email@mail.com"
    then
        GIT_COMMITTER_EMAIL=your_email@mail.com
    fi
' -- --all
```

## Git Large File Storage (LFS)

<https://git-lfs.github.com/>

```
git lfs install` # You only need to run this once per user account
git lfs track "*.psd"`
git add .gitattributes` # Now make sure .gitattributes_ is tracked
git add file.psd`
git commit -m "Add design file"`
git push origin master`
```

## Mensaje de Commit - 'Udacity'

A commit message consists of three distinct parts separated by a blank line:

```
type: subject with 50 characters or less

body is optional and it has 72 characters per line -->

footer is optional
```

The first line is the subject. This should be a short description of what changed. Nothing like “fixed it” or “did something,” these need to be clear and informative, and try to avoid profanity. The subject should be 50 characters or less, with the first letter capitalized, and end without a period.

The type is contained within the title and can be one of these types:

- *feat*: a new feature
- *fix*: a bug fix
- *docs*: changes to documentation
- *style*: formatting, missing semi colons, etc; no code change
- *refactor*: refactoring production code
- *test*: adding tests, refactoring test; no production code change
- *trivial*: updating build tasks, package manager configs, etc; no production code change

The body is next, this is where you give a more detailed description of why you made the change. The body should typically have around 72 characters per line. This is to ensure that the message fits into a terminal window when using git on the command line. You'll also need to make sure there is a blank line between the subject line and the body. You can also add bullet points, using asterisks or dashes, when you need to make a list. Use the body to explain the what and why of a commit, not the how.

Some commits don't require a body in the message. If you fix a typo for example, it's okay to only have a subject line.

You can also include a footer, typically this will be used to indicate which issues or bugs the commit addresses.

#### Example:

```
feat: Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, preceded
  by a single space, with blank lines in between, but conventions
  vary here

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

---

# Github CLI

---

## Configuración

```
# Autenticarse con la cuenta de Github
$ gh auth login

# Mostrar información de la autenticación
$ gh auth status
```

## Mostrar la lista de repositorios

```
gh repo list
```

---

## Enlaces de interés

---

<https://git-scm.com/docs>  
<https://git-scm.com/book/en/v2> | <https://git-scm.com/book/es/v2>  
<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>  
<http://ndpsoftware.com/git-cheatsheet.html>  
<http://ohshitgit.com>  
<https://github.com/arslanbilal/git-cheat-sheet/blob/master/other-sheets/git-cheat-sheet-es.md>  
<https://github.com/git-tips/tips>  
<https://codewords.recurse.com/issues/two/git-from-the-inside-out>  
<https://cli.github.com/>

## Licencia

---



Esta obra está bajo una [licencia de Creative Commons Reconocimiento-Compartir Igual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/).