Comandos de git

1) Primeros comandos

Para empezar con Git, se puede utilizar su sistema de ayuda. Al ejecutar git sin parámetros, se muestra una breve guía con los comandos más comunes y sus descripciones.

☑ **Lista de comandos comunes**:
add, bisect, branch, checkout, clone, commit, diff, fetch, grep, init, log, merge, mv, pull, push, rebase, reset, rm, show, status, tag.
Para obtener ayuda sobre un comando específico, se puede usar **git help <comando>** o **<comando> --help**. (probar cuál funciona, en el entorno donde estamos usando GIT)

☑ **Ejemplo:**
git help init muestra el manual del comando **init**, que crea un repositorio Git vacío o reinicia uno existente.

Nota: En entornos Unix, se puede usar man git-<comando> o man git <comando>, pero este método no funciona en Windows.

Para una lista completa de subcomandos y guías de conceptos, se puede ejecutar:

- **git help -a** para los subcomandos.
- **git help -g para las guías de conceptos, incluyendo "glossary"**.

☑ **Configuración inicial de Git**

Después de instalar Git, se debe realizar una configuración inicial, que se hace **una sola vez** pero puede modificarse después. Esto se logra con el comando **git config**, que establece valores en variables de configuración a nivel de sistema, usuario o proyecto:

A nivel de usuario (--global): para todos los repositorios de ese usuario.

- Ubicación en Windows: C:\Users\Mi usuario\.gitconfig

☑ **Configuración básica:**

- **Identidad del usuario**: es la que haremos en el curso (con esto es suficiente, por supuesto dejamos información adicional, para que descononozcan otros aspectos…)
  - **git config --global user.name "Nombre del Usuario"**
  - **git config --global user.email usuario@ejemplo.com**

Para consultar los valores configurados:

- **git config user.name**
- **git config user.email**
- **git config --list**

Para ver el contenido del archivo de configuración del usuario:

cd ~
cat .gitconfig

2) Conceptos básicos de Git

☺ **Repositorio:** es un **contenedor** que almacena el historial de cambios de un proyecto, registrados a través de commits.

☺ **Commit: guarda** una instantánea del estado del proyecto, incluyendo información sobre el autor, la fecha y una descripción de los cambios, lo que nos permite rastrear modificaciones y autores de cambios en el proyecto.

☺ **Zonas en Git:**

Git utiliza tres zonas locales:

- **Directorio de trabajo (working directory): donde se crean y editan archivos.**
- **Zona de preparación (staging area): donde se guardan las instantáneas de archivos listos para el próximo commit.**
- **Repositorio (repository)**: almacena el historial completo de cambios.

☺ **Estados de un archivo:** Un archivo puede estar en uno de estos estados:

- **Sin modificación**: el contenido es igual en todas las zonas.
- **Modificado**: el contenido difiere entre el directorio de trabajo y las otras zonas.
- **Preparado**: el contenido es igual en la zona de preparación y el directorio de trabajo, pero difiere del repositorio.

☺ **Flujo de trabajo**

☑ Crear un archivo (estado: sin seguimiento).
☑ Añadir el archivo al seguimiento (estado: modificado).
☑ Editar el archivo (estado: modificado).
☑ Preparar el archivo para el commit (estado: preparado).
☑ Hacer commit (estado: sin modificación).

- ☑ Repetir desde el paso 3 para nuevas ediciones.

☺ **SHA-1**
SHA-1 es una **función hash criptográfica** que genera un identificador único de 40 caracteres para cada commit u objeto en Git, permitiendo detectar cualquier cambio en los archivos.

☺ **HEAD**
HEAD es un **puntero** que apunta **al último commit** en la rama activa. Cada commit nuevo actualiza el HEAD para apuntar a este.

☺ **Rama**
Una rama en Git es un **camino alternativo** para el desarrollo de un proyecto, útil para arreglar errores o probar ideas nuevas. **La rama por defecto es "master"**, pero se pueden crear más y cambiar entre ellas, afectando el contenido del directorio de trabajo.
Por ejemplo:

- ☑ Se crean dos commits (75528b9 y 8daf16a) en la rama master.
- ☑ Se crea una nueva rama "pruebas" con dos commits nuevos (a3ae45c y 456af81).
- ☑ Se vuelve a la rama master y se introduce un nuevo commit (de396a3).
- ☑ HEAD apunta al último commit de la rama activa.

3) Inicializando un Repositorio

Para trabajar con Git, primero hay que inicializar el repositorio, para ello, tenemos 3 opciones:

Opción 1: Clonando un Repositorio Existente:

- ☑ Se usa git clone URL [directorio] para traer el repositorio remoto a nuestro equipo.
- ☑ Ejemplo: git clone https://github.com/jquery/jquery.git
- ☑ Esto descarga todos los commits y archivos del proyecto.

Opción 2: A Partir de un Proyecto Existente:

- ☑ Se crea un directorio nuevo y se inicializa el repositorio con git init.
- ☑ Luego se pueden crear archivos en este directorio y añadirlos al control de versiones.

Opción 3: A Partir de un Proyecto Nuevo:

- ☑ Similar al anterior, pero se parte de un directorio vacío.
- ☑ Se crea el directorio, se inicializa el repositorio y se añade contenido nuevo.

En todos los casos, el directorio .git contiene toda la información del repositorio, y git status permite ver el estado de los archivos.

4) Resumen de los pasos para añadir un archivo a un repositorio Git

- ☑ **Crear un archivo nuevo:**

`touch archivo_a.txt`

☑ **Verificar el espacio de trabajo:**
`ls -la`
se verá el nuevo archivo archivo_a.txt.

☑ **Verificar el estado del repositorio:**
`git status`
Se muestra archivo_a.txt como no seguido (untracked).

☑ **Agregar el archivo a la zona de preparación:**
`git add archivo_a.txt`

☑ **Para agregar todos los archivos no seguidos:**
`git add`

☑ **Verificar el estado del repositorio nuevamente:**
`git status`
Se mostrará archivo_a.txt como preparado para ser commit.

☑ **Hacer el primer commit:**
`git commit -m "Añado el archivo_a.txt vacío"`

☑ **Verificar el estado del repositorio después del commit:**
`git status`
El mensaje indicará que el directorio de trabajo está limpio (sincronizado).

5) Añadir y confirmar más archivos

☑ **Crear nuevos archivos:**
`touch archivo_b.txt`
`touch archivo_c.txt`

☑ **Verificar el estado del repositorio:**
`git status`
Muestra los nuevos archivos como no seguidos.

☑ **Agregar archivo_b.txt a la zona de preparación:**
`git add archivo_b.txt`

☑ **Confirmar el archivo archivo_b.txt:**
`git commit -m` "Añado el archivo_b.txt vacío"

☑ **Agregar y confirmar archivo_c.txt:**
`git add .`
`git commit -m` "Añado el archivo_c.txt vacío"

6) Editar archivos y realizar commits

☑ **Editar un archivo:**
`echo "Creo una primera línea en archivo_a.txt" >> archivo_a.txt`

☑ **Verificar el estado del repositorio:**
`git status`
archivo_a.txt se muestra como modificado.

☑ **Agregar el archivo editado a la zona de preparación:**

```
git add archivo_a.txt
```

☑ **Confirmar el archivo editado:**

```
git commit -m "Introduzco una línea en archivo_a.txt"
```

☑ **Editar y agregar más archivos:**

```
echo "Creo una primera línea en archivo_b.txt" >> archivo_b.txt
echo "Creo una primera línea en archivo_c.txt" >> archivo_c.txt
git add archivo_b.txt archivo_c.txt
git commit -m "Introduzco líneas en archivos_b.txt y
archivo_c.txt"
```

7) Visualizar el historial de commits

☑ **Ver el historial de commits en una línea:**

```
git log --oneline
```

Muestra una lista de los commits realizados.

Estos son los pasos básicos para añadir, editar y confirmar archivos en un repositorio Git, además de ver el historial de commits.

8) Modificar commits

☑ **Importancia del SHA-1**

El SHA-1 de un commit depende de los predecesores, lo que significa que modificar un commit cambia su SHA-1 y el de todos los commits posteriores. Así, solo se puede modificar el último commit sin reescribir la historia de todo el repositorio.

☑ **Visualización de commits** para ver los últimos commits:

```
git log --oneline -4
```

Esto muestra algo como:

```
css
Copiar código
fd79ed5 Añado el archivo .gitignore y el archivo imagenes/logo.png
44a9a8c Renombramos tres archivos
63a453b Añado tres archivos de prueba para moverlos
b5b78cc Borro el archivo temporal_2.txt
```

☑ **Modificar el último commit**

- **Añadir una línea a .gitignore:**

```
echo "ejecutables/*.exe" >> .gitignore
```

- **Verificar cambios:**

```
git status
```

Esto mostrará:

```
makefile
modified: .gitignore
```

- **Añadir y hacer un nuevo commit con --amend:**

```
git add .gitignore
```

git commit --amend -m "Añado los archivos .gitignore e imagenes/logo.png"
Esto cambia el SHA-1 y el mensaje del último commit.

- **Verificar historial:**
  git log --oneline -4
  El nuevo SHA-1 será diferente, por ejemplo:
  css
  f00704b Añado los archivos .gitignore e imagenes/logo.png
  44a9a8c Renombramos tres archivos
  63a453b Añado tres archivos de prueba para moverlos
  b5b78cc Borro el archivo temporal_2.txt
- **Ver contenido de .gitignore**
  cat .gitignore
  Mostrará el contenido actualizado, incluyendo:
  ejecutables/*.exe

9) Revertir un commit

☑ **Crear un nuevo commit para revertir** Para revertir el último commit
git revert f00704b

Esto genera un nuevo commit que deshace los cambios del commit f00704b.

☑ **Visualizar el nuevo commit**
git log --oneline -4
Mostrará:
css
Copiar código
7c51cef Revert "Añado los archivos .gitignore e imagenes/logo.png"
f00704b Añado los archivos .gitignore e imagenes/logo.png
44a9a8c Renombramos tres archivos
63a453b Añado tres archivos de prueba para moverlos

☑ **Ver estado del repositorio**
git status
Mostrará archivos sin seguimiento debido a la eliminación de .gitignore:
compilados/
imagenes/
log/
temporal_6.txt
temporal_7.zip

10) Eliminar archivos no seguidos

Para limpiar archivos no seguidos:
git clean -f

Esto eliminará archivos sueltos. Para eliminar directorios y sus contenidos:

git clean -f -d

Esto elimina todos los archivos y directorios no seguidos, dejando el repositorio limpio.

11) Recuperar un archivo de un commit

☑ Para recuperar .gitignore del commit f00704b:

git checkout f00704b -- .gitignore

Esto coloca el archivo en la zona de preparación.

☑ **Finalizar con un commit**

git commit -am "Añado el archivo .gitignore"

Esto genera un nuevo commit con el archivo recuperado.

12) Revertir commits con reset

☑ **Hard reset**

Para resetear el repositorio, zona de preparación y zona de trabajo:

git reset --hard f00704b

Esto deja todo en el estado del commit f00704b.

☑ **Mixed reset**

Para resetear solo el repositorio y la zona de preparación:

git reset --mixed f00704b

Deja los cambios en la zona de trabajo sin preparar.

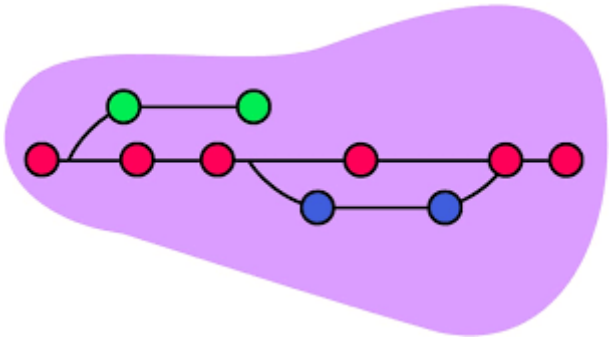
☑ **Soft reset**

Para resetear solo el repositorio:

git reset --soft f00704b

Deja los cambios preparados para un nuevo commit.

13) Ramas



☺ Las ramas **son bifurcaciones o caminos alternativos** que se siguen en un proyecto de software, desarrollo web o diseño.

☺ Git crea una **rama principal** llamada **"master" por defecto**, pero se pueden crear más ramas y cambiar entre ellas.

☺ Las razones para usar una rama son tener una versión estable en la rama "master" y otra rama de "desarrollo", probar ideas que pueden ser descartadas o llevadas a la rama principal en el futuro, y arreglar errores en una rama específica antes de llevar los cambios a la rama principal.

¿Cómo trabajamos con las ramas?

☑ **Para crear una rama,**
git branch [nombre_de_la_rama]

☑ **para cambiar a una rama se utiliza**
git checkout [nombre_de_la_rama]

☑ **Para crear y cambiar a una rama en un solo paso**
git checkout -b [nombre_de_la_rama]

Al trabajar con una rama, los cambios y commits se realizan en la rama activa, y al cambiar de rama, el contenido del espacio de trabajo cambia acorde a los cambios y commits en la rama activa.

☑ **Para mostrar las ramas**
git branch

☑ **para cambiar de rama se utiliza**
git checkout [nombre_de_la_rama]

Al trabajar con varias ramas, es importante tener en cuenta en qué rama se están realizando los cambios y commits..

☑ **Renombrar una rama**
git branch -m [nombre_actual] [nombre_nuevo]
Por ejemplo, para renombrar la rama "experimento" a "intento":
git branch -m experimento intento

☑ **Borrar una rama**
git branch -d [nombre_de_la_rama]
Si estás en la rama que deseas borrar, necesitas cambiar primero de rama:
git checkout master
git branch -d rama_temporal

☑ **Fusionar ramas**
Para fusionar los cambios de una rama en otra, se emplea
git merge [nombre_de_la_rama]
Por ejemplo, para fusionar la rama "intento" en "master":
git checkout master

git merge intent

- ☑ **Conflictos en la fusión entre ramas**

Cuando hay conflictos al fusionar ramas, Git muestra un mensaje de conflicto en los archivos afectados. Debes resolver manualmente estos conflictos, eliminando las marcas **<<<<<<<, =======, y >>>>>>>.**

- ☑ **Rebase:** (**git rebase [rama_base]**) es otra forma de integrar cambios de una rama en otra, reorganizando el historial de commits de forma lineal.
- ☑ **Stash:** (**git stash**) permite guardar temporalmente los cambios en el directorio de trabajo sin necesidad de hacer commit, útil cuando necesitas cambiar de rama rápidamente. Estos comandos y técnicas son fundamentales para gestionar eficientemente las ramas en Git.

14) Historial

- ☑ **Clonar el proyecto** gitignore desde GitHub
  git clone https://github.com/github/gitignore.git
- ☑ **Verificar el último commit** realizado en el repositorio
  git log -n 1
- ☑ **Revisar los últimos 2 commits** en el repositorio
  git log -2
- ☑ **Buscar los commits realizados por "Gary Smith**"
  git log --author="Gary Smith"
- ☑ **Obtener los commits de "Gary Smith" mostrados de forma resumida**:
  git log --oneline --author="Gary Smith"
- ☑ **Buscar los commits desde el 1 de octubre de 2015**
  git log --since="2015-10-01" --oneline
- ☑ **Filtrar los commits entre el 1 y el 10 de octubre de 2015**
  git log --since="2015-10-01" --until="2015-10-10" --oneline
- ☑ **Buscar los commits realizados hace 3 semanas hasta hace 2 semanas:**
  git log --since="3 weeks ago" --until="2 weeks ago" --oneline
- ☑ **Comandos similares** con formas abreviadas para obtener los mismos resultados:
  git log --since=3.weeks --until=2.weeks --oneline
- ☑ **Buscar commits que mencionaran la palabra "Emacs"**
  git log --grep="Emacs" --oneline
- ☑ **Comparar los cambios entre dos commits** específicos
  git log 4892e96..d06a6e2 --oneline
- ☑ **Buscar commits que afectaran al archivo ".gitignore"**
  git log --oneline .gitignore
- ☑ **Revisar el historial de commits del archivo ".gitignore" con detalles de cambios**

```
git log -p .gitignore
```

☑ **Visualizar el historial de commits de todas las ramas en formato gráfico**

```
git log --oneline --graph --all --decorate
```

☑ **Obtener estadísticas de modificaciones por commit**

```
git log --stat --oneline
```

15) Cambios en el proyecto

☑ Para revisar diferencias entre dos commits, utilicé git diff:

```
git diff
```

☑ Creé un nuevo directorio y configuré un repositorio Git:

```
cd ~/proyectos
mkdir cambios
cd cambios
git init
```

☑ Realicé cambios en varios archivos y usé git status para verificar el estado:

```
echo "Línea 1 archivo_a.txt" >> archivo_a.txt
git add .
git commit -m "Línea 1 al archivo_a.txt"

echo "Línea 2 archivo_a.txt" >> archivo_a.txt
echo "Línea 1 archivo_b.txt" >> archivo_b.txt
git add .
git commit -m "Línea 1 al archivo_b.txt y línea 2 al archivo_a.txt"

echo "Línea 3 archivo_a.txt" >> archivo_a.txt
echo "Línea 2 archivo_b.txt" >> archivo_b.txt
echo "Línea 1 archivo_c.txt" >> archivo_c.txt
```

☑ Verifiqué el estado actual con git status:

```
git status
```

☑ Mostré las diferencias entre el área de preparación (staged) y el último commit:

```
git diff --staged
```

☑ Realicé un commit con los cambios preparados:

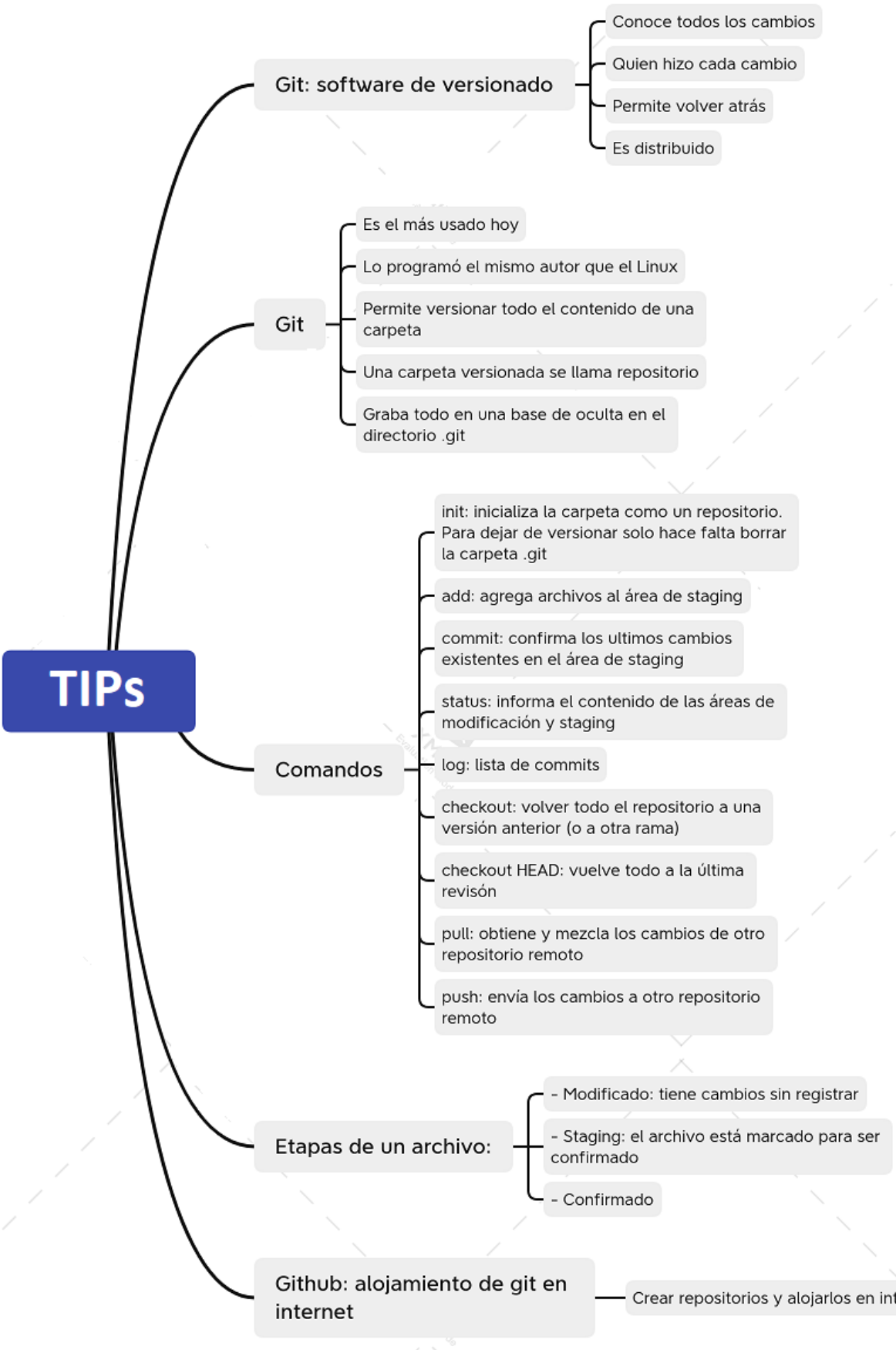
```
git commit -m "Línea 1 archivo_c.txt, línea 2 archivo_b.txt,
línea 3 archivo_a.txt"
```

☑ Verifiqué nuevamente el estado y los cambios con git status y git diff:

```
git status
git diff
git diff --staged
```

☑ Finalmente, revisé el historial de commits para confirmar los cambios realizados:

```
git log --oneline
```


TIPs
Git: software de versionado
Conoce todos los cambios
Quien hizo cada cambio
Permite volver atrás
Es distribuido
Git
Es el más usado hoy
Lo programó el mismo autor que el Linux
Permite versionar todo el contenido de una carpeta
Una carpeta versionada se llama repositorio
Graba todo en una base de oculta en el directorio .git
Comandos
init: inicializa la carpeta como un repositorio. Para dejar de versionar solo hace falta borrar la carpeta .git
add: agrega archivos al área de staging
commit: confirma los ultimos cambios existentes en el área de staging
status: informa el contenido de las áreas de modificación y staging
log: lista de commits
checkout: volver todo el repositorio a una versión anterior (o a otra rama)
checkout HEAD: vuelve todo a la última revisón
pull: obtiene y mezcla los cambios de otro repositorio remoto
push: envía los cambios a otro repositorio remoto
Etapas de un archivo:
- Modificado: tiene cambios sin registrar
- Staging: el archivo está marcado para ser confirmado
- Confirmado
Github: alojamiento de git en internet
Crear repositorios y alojarlos en internet