

GIT: INTRODUCCIÓN

A series of horizontal lines in teal and light blue colors, spanning the width of the slide and partially overlapping the white content area.

1r DAW

Jose Manuel Barrera Arroyo

QUE ES GIT








- Sistema de control de versiones (SCV) o (VCS en inglés)
- Se trata de un repositorio DISTRIBUIDO
 - ¿Qué ventajas presenta?
- Facilita el trabajo en grupo
- En abierto
- Permite tener un historial y una trazabilidad de cada uno de los cambios en un proyecto

QUE ES GIT

- Inventado por Linus Torvalds
- Código en abierto (se puede ver el código en la propia página web, que usa GIT)
- No es el único SCV (Monotone, Mercurial, Atlassian, etc)
- Si sabes usar uno, sabes usar prácticamente todos
- **¡Para código textual, no código compilado!**

QUE ES GIT

- Antes

Nombre	Estado
 ProyectoFinal.txt	✓
 ProyectoFinalDeVerdad.txt	✓
 ProyectoFinalDeVerdadDefinitivo.txt	✓
 ProyectoFinalV02.txt	✓
 ProyectoV01.txt	✓
 ProyectoV02.txt	✓
 ProyectoV03.txt	✓



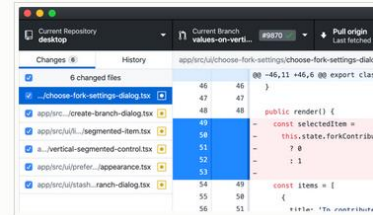
QUE ES GIT

- Ahora



INTERFACES DE GIT

- Hay muchas
- Pero la de verdad, la de machotes, por consola (bash)
- Donde más se aprende
- Nosotros usaremos GIT Kraken como GUI (Guide User Interface)



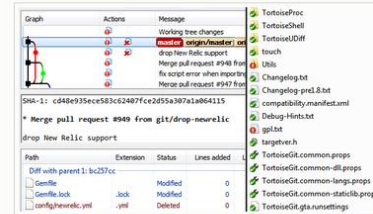
GitHub Desktop

Platforms: Mac, Windows
Price: Free
License: MIT



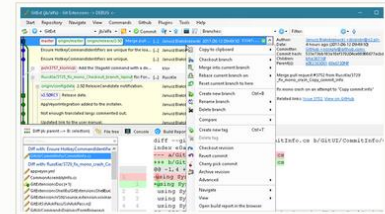
SourceTree

Platforms: Mac, Windows
Price: Free
License: Proprietary



TortoiseGit

Platforms: Windows
Price: Free
License: GNU GPL



Git Extensions

Platforms: Windows
Price: Free
License: GNU GPL

INTERFACES DE GIT

- Vamos a hacer todo por duplicado:
 - Con GUI
 - Con consola
- GUI es más cómodo, pero la consola es muuuuuucho más potente

INTERFACES DE GIT


- Más de 1000 comandos
- Con 10, ya puedes hacer el 95% de las cosas
- **GIT es básico para programar, si lo conocéis y sabéis operar con él, tenéis faena.**

INICIO: Usuario

- GIT lleva un férreo control de qué hace qué cosa.
- Solo se necesitan dos cosas para poder empezar a operar con GIT
 - **Usuario**
 - **Email**

INICIO: Usuario

- GIT Kraken
 - En el momento que entras a GIT Kraken, la primera cosa que te pide.
- BASH
 - `git config --global user.name "Full name"`
 - `git config --global user.email lo@quesea.com`



Ojo, si ya tienes cuenta en github, el mail debería de ser el mismo que el que tienes dentro de github. Si aún no has creado tu cuenta de github, no hay ningún problema. Cuando ya la crearás y deberás de usar este mail.

INICIO: Creación directorio

- Un repositorio se puede:
 - Crear directamente bajo CV (control de versiones)
 - Convertir un directorio existente a CV

INICIO: Creación directorio

- GIT Kraken

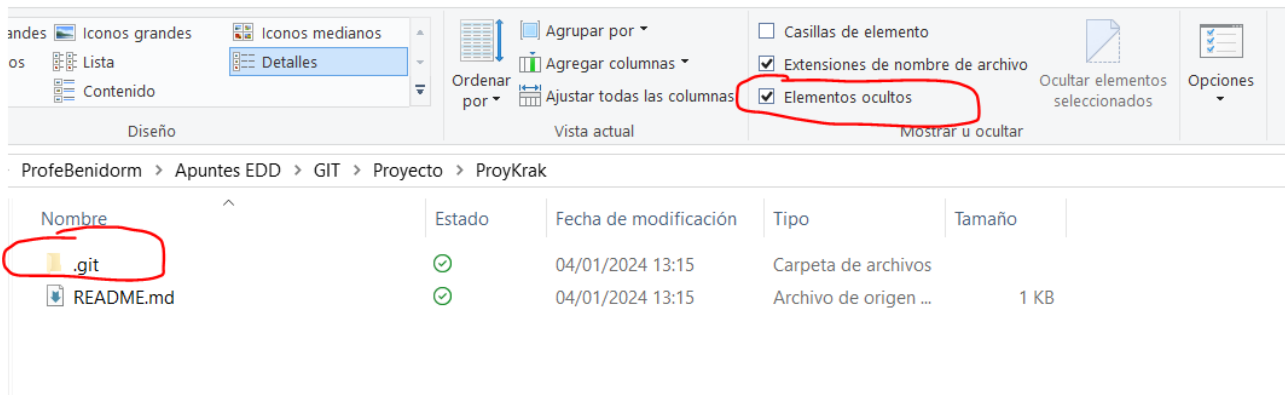
- File → Init Repo → Seleccionar directorio → Dar nuevo nombre al proyecto (y se creará directorio específico bajo SCV)

- BASH

- Me posiciono en el directorio que quiero:
 - `git init`
- O si estoy en un directorio, y quiero crear el proyecto “blabla” bajo control de versiones (similar a como lo hace GitKraken)
 - `git init blabla`

INICIO: Creación directorio

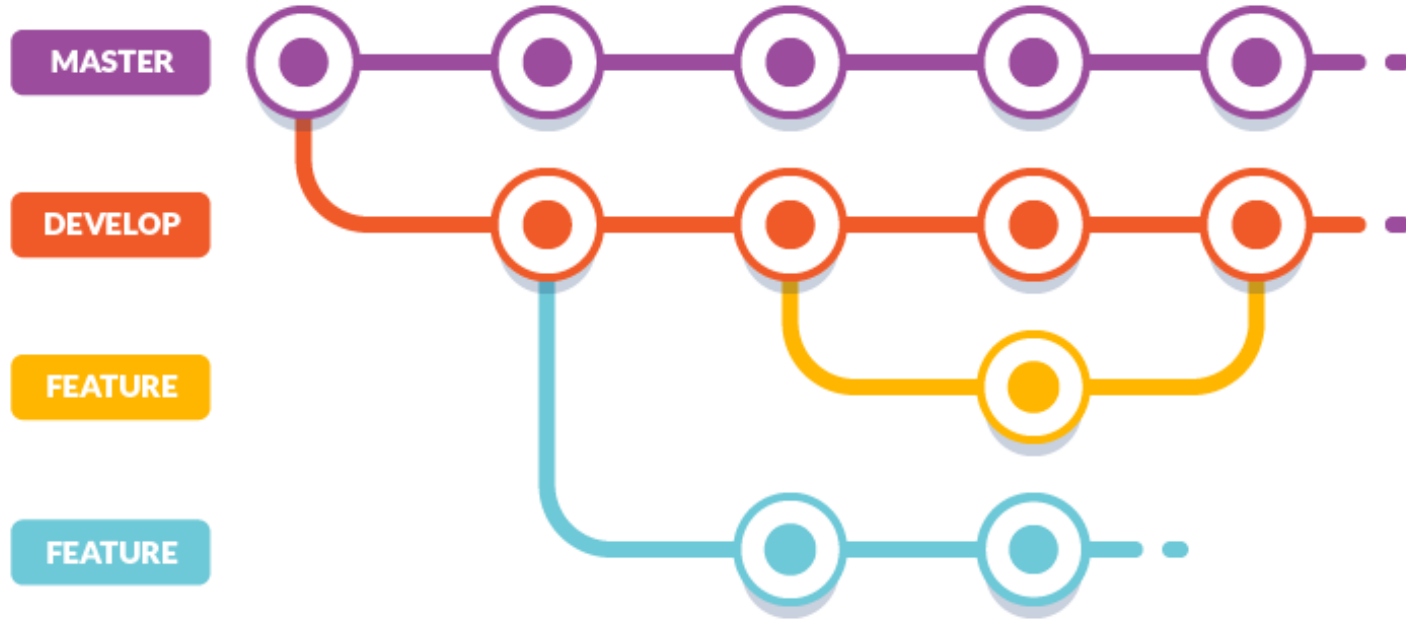
- Al inicializar un repositorio GIT, se crea una carpeta oculta .git



INICIO: RAMAS

- Cuando se crea el directorio, se parte de la rama principal (antes se llamaba “master”, ahora “main”)
- El concepto de ramas es lo que hace a GIT tan potente
- Las ramas siguen una estructura jerárquica:
 - Cuánto más alejada de la rama principal, más se trabaja en ellas y más en desarrollo está el programa
 - Cuánto más cercana a la rama principal, menos tiempo se está en ella, y más definitivo es el programa

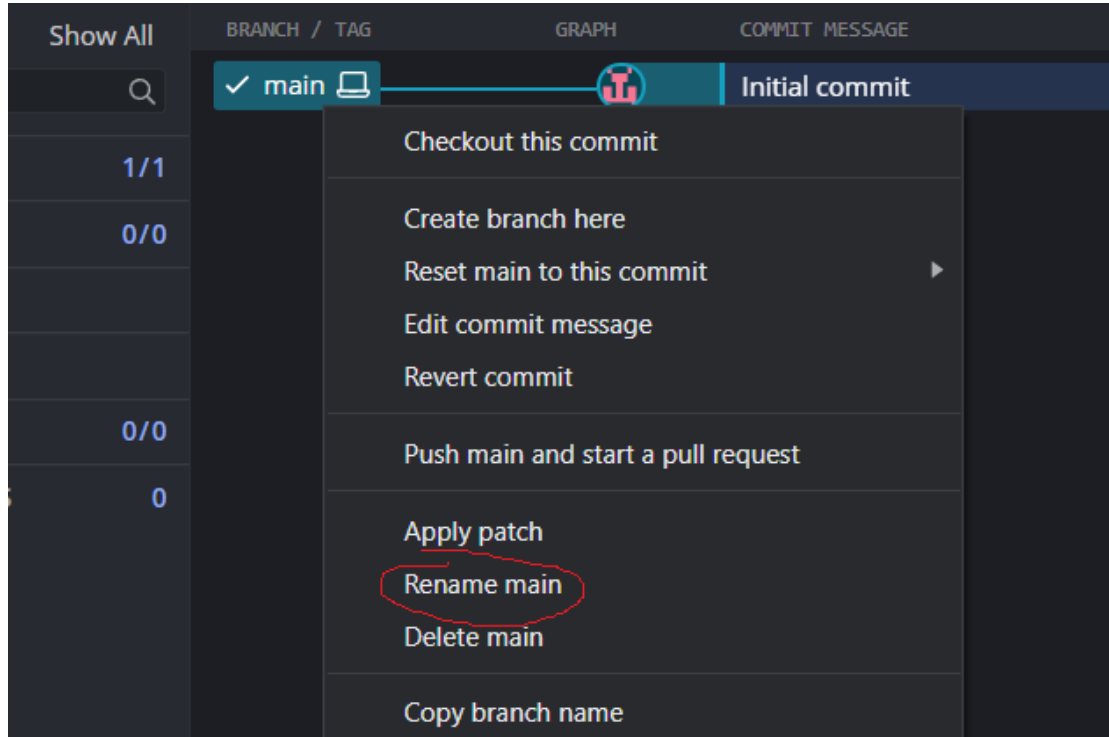
INICIO: RAMAS



INICIO: RAMAS

- Se puede cambiar el nombre de la rama (si os hace ilusión)
- GIT Kraken
 - Click derecho en la rama → Rename main

INICIO: RAMAS



INICIO: RAMAS

- Bash:
 - En el directorio:
 - `git branch -m nuevonombre`
 - Podemos comprobar el cambio con:
 - `git branch`

INICIO: ESTADO

- Un directorio bajo CV funciona haciendo snapshots del proyecto
- Nosotros elegimos cuando queremos hacer una snapshot
- Podemos hacer un montón de cosas, y querer hacer una snapshot, o hacer muchas snapshots de pequeñas cosas
- En el mundo real se realizan las snapshots cuando se ha implementado una funcionalidad

INICIO: ESTADO

- Bash:

- Si no hemos hecho nada, y hacemos:

- Git status

```
PS C:\Users\JOSEMANUEL BARRERAARR\OneDrive  
s EDD\GIT\Proyecto\Proy1> git status  
On branch master  
nothing to commit, working tree clean
```

- Nos dice la rama en la que estamos, y que no hay nada para hacer un commit (ya veremos lo que es)

INICIO: ESTADO

- Bash:
 - Si no hemos hecho nada, y hacemos:
 - Git status
- Nos dice la rama en la que estamos, y que no hay nada para hacer un commit (ya veremos lo que es)

```
PS C:\Users\JOSEMANUEL BARRERAARR\OneDrive  
s EDD\GIT\Proyecto\Proy1> git status  
On branch master  
nothing to commit, working tree clean
```

INICIO: ESTADO

- Bash:
 - En el momento que añade un archivo (nosotros trabajaríamos igual que siempre), y hago el “git status”

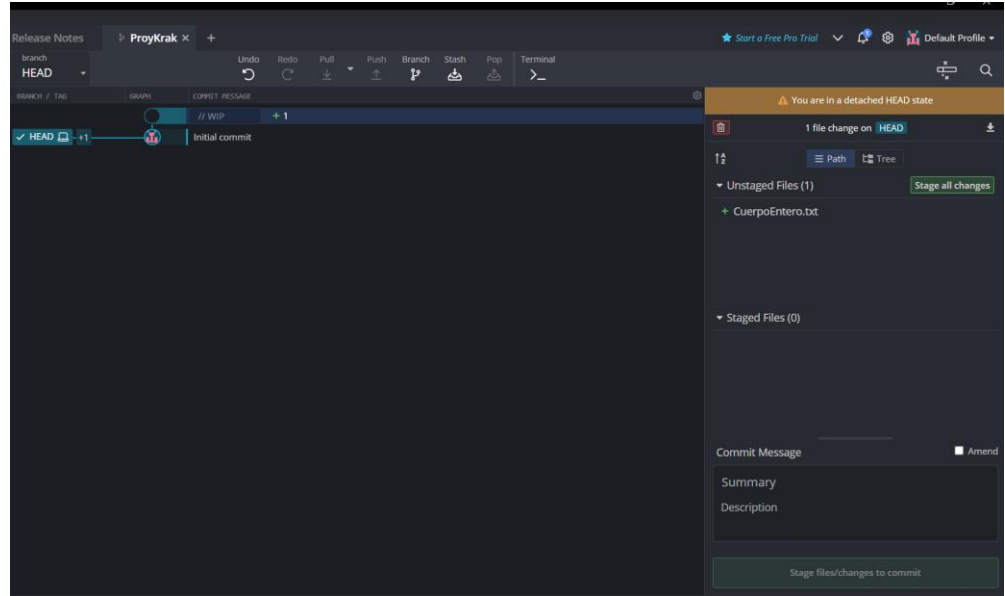
```
s git statusoyecto\Proy1>  
On branch master  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    CuerpoEntero.txt
```

INICIO: ESTADO

- Bash:
 - Me dice la rama, y que tengo cambios respecto al último snapshot que se ha hecho, o lo que es lo mismo, tango cambios en los cuáles no he hecho ningún snapshot

INICIO: ESTADO

- GIT Kraken:
 - Te avisa el programa en la parte de la derecha



OPERANDO: PREPARAR SNAPSHOT

- El siguiente paso es añadir los archivos de los cuáles voy a trackear (ahora están en untracked)
- OJO, con el siguiente comando, aún no se ha hecho la snapshot, si no que se preparan para ello, es un paso intermedio

OPERANDO: PREPARAR SNAPSHOT

- BASH
 - `git add Nombredelarchivo.conextension`
 - Esto hace que el archivo pase a la fase preparación de snapshot, si luego hacemos el `git status`, vemos que ahora ha cambiado

OPERANDO: PREPARAR SNAPSHOT

- BASH

```
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD  
s EDD\GIT\Proyecto\Proy1> git add CuerpoEntero.txt  
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD  
s EDD\GIT\Proyecto\Proy1> git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    new file:   CuerpoEntero.txt
```

- Al hacer el git add, vemos que me dice que hay cambios preparados para hacer la snapshot (el commit)

OPERANDO: PREPARAR SNAPSHOT

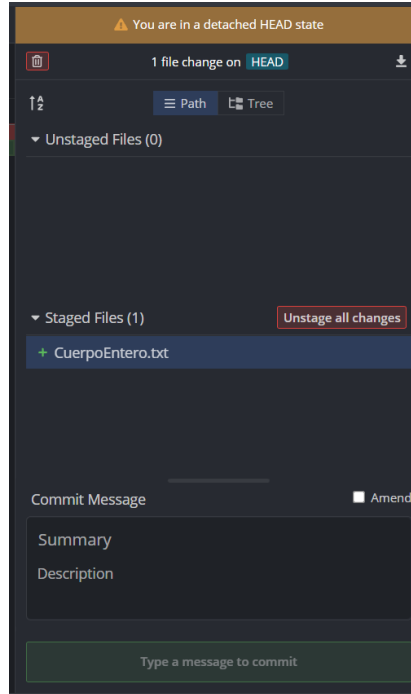
- BASH
 - Lo normal es usar el comando
 - `git add .`
 - Este comando prepara todos los cambios del directorio, no va archivo por archivo

OPERANDO: PREPARAR SNAPSHOT

- GIT Kraken
 - Seleccionar el archivo que quieres stagear
 - Darle a “Stage File”
 - Veremos que el archivo ha pasado a Staged Files (podríamos volver a pasarlo a Unstaged, si quisiésemos)
 - Tenemos la opción de “Stage all files” que equivale al git add .

OPERANDO: PREPARAR SNAPSHOT

- GIT Kraken

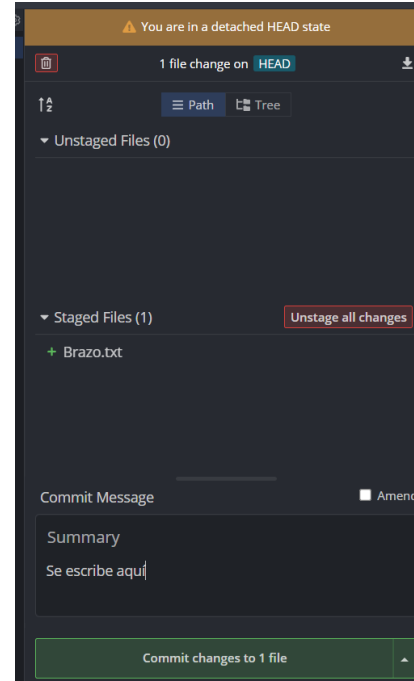


OPERANDO: HACER LA SNAPSHOT

- Creamos ya una snapshot
- Tiene un código hash asociado irrepetible, que nos permite volver en algún momento a ese estado en concreto
- Obliga a poner un mensaje siempre, para saber que hemos hecho

OPERANDO: HACER LA SNAPSHOT

- GIT Kraken
 - Escribir un mensaje el recuadro, y darle a enviar.
 - No te deja enviar, si no hay un comentario



OPERANDO: HACER LA SNAPSHOT

- Bash
 - `git commit -m "Mensaje de commit"`

```
PS C:\Users\JOSEMANUEL BARRERAARR\OneDrive - UNIVERSIDAD  
s EDD\GIT\Proyecto\Proy1> git commit -m "Primer commit"  
[master 2df8929] Primer commit  
1 file changed, 1 insertion(+)  
create mode 100644 CuerpoEntero.txt  
PS C:\Users\JOSEMANUEL BARRERAARR\OneDrive - UNIVERSIDAD  
s EDD\GIT\Proyecto\Proy1> git status  
On branch master  
nothing to commit, working tree clean
```

OPERANDO: HACER LA SNAPSHOT

- Si después de hacer un commit, hacemos un “git status”, veremos que nos dice que nuestro directorio coincide con el que está almacenado
 - git log
 - Nos da información de los commits, con su hash

```
PS C:\Users\JOSEMANUEL BARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Es  
s EDD\GIT\Proyecto\Proy1> git log  
commit 2df8929af2a8db95b7425deda0d908bca2840f3a (HEAD -> master)  
Author: Jose Manuel Barrera <jm.barreraarroyo@edu.gva.es>  
Date: Thu Jan 4 15:00:26 2024 +0100  
  
Primer commit  
  
commit 483521579f9d32c4929fb7888509399cb98566cd  
Author: Jose Manuel Barrera <jm.barreraarroyo@edu.gva.es>  
Date: Thu Jan 4 13:24:36 2024 +0100  
  
Initial commit
```

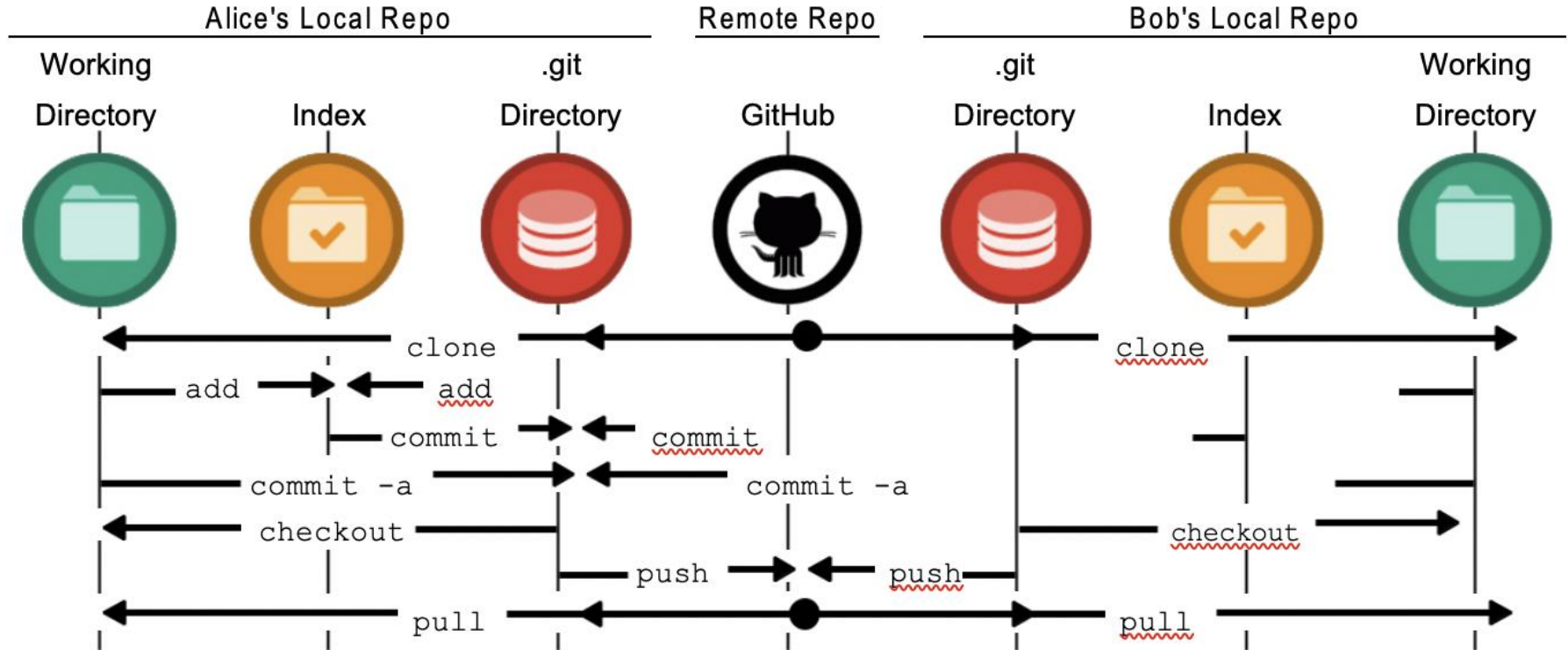
OPERANDO: RESETEAR CAMBIOS

- Si después de modificar cosas, queremos volver al último estado que hemos guardado
 - `git reset`

OPERANDO: FLUJO DE TRABAJO



OPERANDO: FLUJO DE TRABAJO



OPERANDO: VISUALIZAR RAMA

- Existen gran cantidad de comandos que nos permiten configurar como visualizar la rama.
- El punto de partida es “git log”, pero hay otras muchas opciones.

OPERANDO: VISUALIZAR RAMA

- `git log --graph`
 - Ya no enumera únicamente, si no que muestra las ramas

```
> git log --graph
* commit cac4e795d5352340fd20026ce7c9525c89a80641 (HEAD -> master)
  Author: Jose Manuel Barrera <jm.barreraarroyo@edu.gva.es>
  Date: Mon Jan 15 12:43:41 2024 +0100

    Añadimos todo junto

* commit 2df8929af2a8db95b7425deda0d908bca2840f3a
  Author: Jose Manuel Barrera <jm.barreraarroyo@edu.gva.es>
  Date: Thu Jan 4 15:00:26 2024 +0100

    Primer commit

* commit 483521579f9d32c4929fb7888509399cb98566cd
  Author: Jose Manuel Barrera <jm.barreraarroyo@edu.gva.es>
  Date: Thu Jan 4 13:24:36 2024 +0100

    Initial commit
```

OPERANDO: VISUALIZAR RAMA

- `git log --graph --pretty=oneline`
 - Reduce la información por commit

```
> git log --graph --pretty=oneline
* cac4e795d5352340fd20026ce7c9525c89a80641 (HEAD -> master) Añadimos todo junto
* 2df8929af2a8db95b7425deda0d908bca2840f3a Primer commit
* 483521579f9d32c4929fb7888509399cb98566cd Initial commit
```


OPERANDO: VISUALIZAR RAMA

- `git log --decorate --all --oneline`
 - Reduce la información por commit, y también la del hash

```
> git log --graph --decorate --all --oneline
* cac4e79 (HEAD -> master) Añadimos todo junto
* 2df8929 Primer commit
* 4835215 Initial commit
```

OPERANDO: ALIAS

- Como es un peñazo aprenderse configuraciones especiales de un comando, podemos agregarlas bajo un alias, de manera que cuando se llame a ese alias, se ejecute el comando especial de configuración
- Esto se hace modificando el archivo de configuración
 - `git config --global alias.nombrequeledoy "todo el comando"`
 - Ejemplo: `git config --global alias.arbolreduct "log --decorate --all --oneline"`

OPERANDO: ALIAS

- Ojo! Si da un error, probar a añadir el símbolo de exclamación invertido en la creación del alias
 - `git config --global alias.nombrequeledoy "!todo el comando"`
 - Ejemplo: `git config --global alias.arbolreduct "!git log -decorate -all --oneline"`

OPERANDO: GITIGNORE

- Muchas veces, yo quiero que un archivo esté fuera del SCV.
 - Por ejemplo, un archivo de sistema en el que aparezca la ruta a un archivo mío.
 - Porque seguramente la persona que acceda al archivo tendrá la información en otro proyecto en otra ruta completamente distinto.
 - También puede ser un problema en archivos compilados en código binario, que usen variables específicas de sistema.

OPERANDO: GITIGNORE

- Para ello, debo de crear un **ARCHIVO** que **OBLIGATORIAMENTE** debe de tener el nombre de .gitignore
- No es un archivo de texto, y no tiene que tener extensión
- Todo nombre de archivo o carpeta que escribamos dentro, no será tomada en cuenta para la operación de GIT

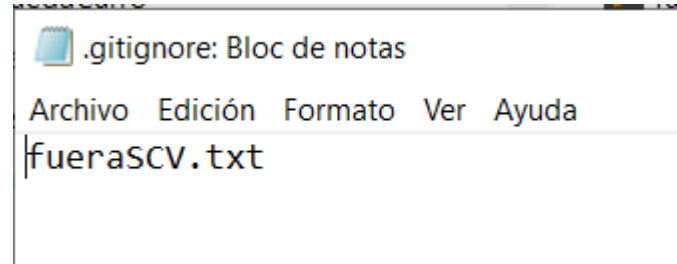
OPERANDO: GITIGNORE

- Por ejemplo, creo un documento de texto, que quiero tener fuera de SCV.

Nombre	Estado	Fecha de modificación
 .git		15/01/2024 16:36
 .gitignore		15/01/2024 16:35
 CuerpoEntero.txt		05/01/2024 13:04
 fueraSCV.txt		15/01/2024 16:35
 README.md		04/01/2024 12:50

OPERANDO: GITIGNORE

- Para ello, creo el .gitignore, y añado el nombre del archivo (fueraSCV.txt)



OPERANDO: GITIGNORE

- Si el archivo se crea antes de meterlo en el .gitignore y se trackea, se puede quedar en la memoria caché de git, y lo detectará siempre como trackeado.
- Por eso es mejor añadir al gitignore primero, y luego crear el archivo. Si no se puede, se puede limpiar la caché con:
 - `git rm --cached nombredelarchivo`
- Cuando intento añadir los archivos al staging area, ya no detecta el archivo fueraSCV.txt, pero si el gitignore, que obviamente, debo de añadir.

OPERANDO: GITIGNORE

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  fueraSCV.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\Pr
> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\Pr
> git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
  add
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\Pr
> git add .
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\Pr
> git commit -m "Se añade el gitignore"
[master 7c60d33] Se añade el gitignore
1 file changed, 1 insertion(+)
create mode 100644 .gitignore
```

Momento de
creación de
.gitignore

OPERANDO: GIT DIFF

- Es posible ver, la diferencia que tengo (con mucho detalle) desde el momento actual, a la última snapshot, mediante el comando:
 - `git diff`

OPERANDO: GIT DIFF

```
> git diff
diff --git a/CuerpoEntero.txt b/CuerpoEntero.txt
index 47019b6..7bbd27e 100644
--- a/CuerpoEntero.txt
+++ b/CuerpoEntero.txt
@@ -1,3 +1,5 @@
-Cuerpo del potato
+Cuerpo del potato podrido

-600890634
\ No newline at end of file
+600890634
+
+Añado una línea
```