**Software Libre y Compromiso Social**
**Práctica 2: Git**

## Introducción a Git

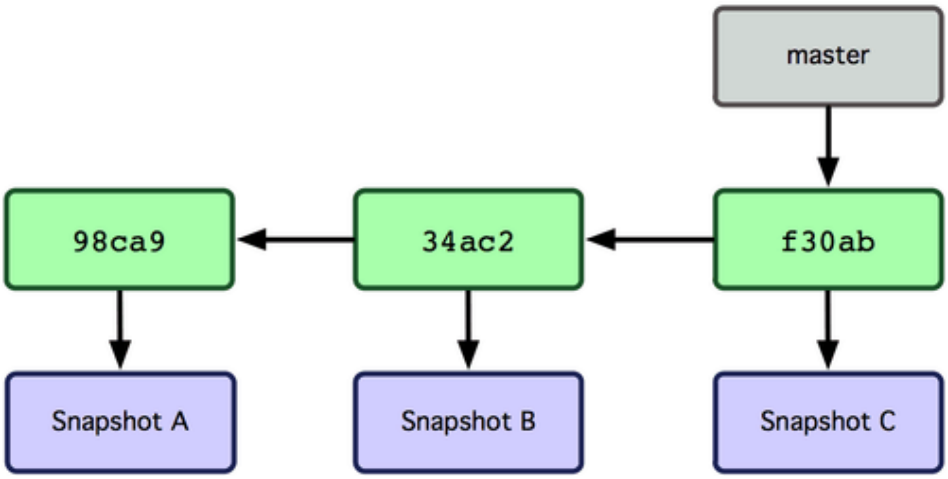
Git (pronunciado "Guit") es un software de control de versiones diseñado por *Linus Torvalds* pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en sucesivas versiones de los archivos y coordinar el trabajo y los cambios que varias personas realizan cuando se trata de un proyecto compartido. Git tiene licencia GPL.

Características principales destacables:

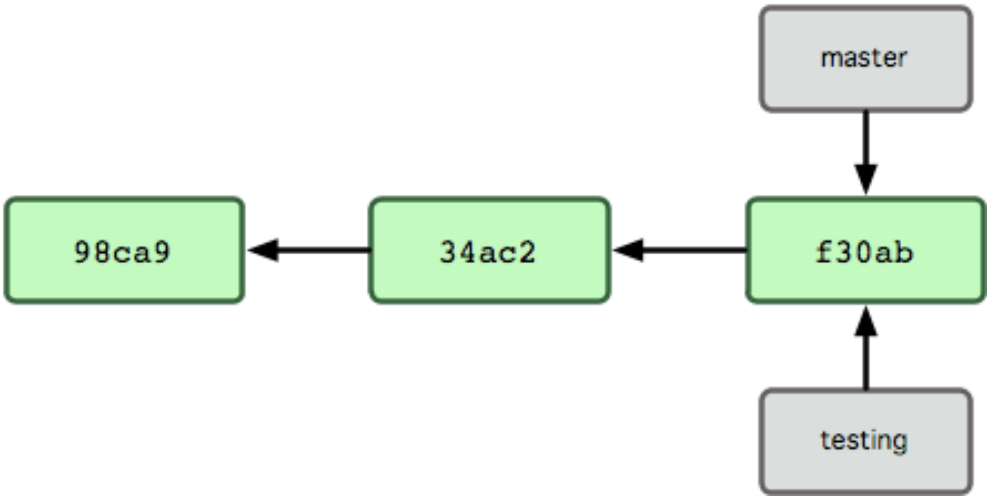
- No hay una localización centralizada de los datos compartidos. El proyecto o repositorio está o puede estar **realmente distribuido**, se puede trabajar offline independientemente en copias distribuidas localmente y posteriormente integrarlas.
- Hay control de integridad. Si se corrompe algún fichero por cualquier motivo, Git lo detecta y te ayuda a solucionarlo. Si pones algo en Git, está garantizado que cuando lo sacas su contenido es exactamente el mismo, **sin corrupción o pérdidas de información** de ningún tipo. Si recuperamos una revisión antigua del proyecto, podemos estar seguros que nadie ha cambiado ni un solo bit de ningún archivo: Git se daría cuenta. Esto es especialmente importante en un sistema distribuido con muchos participantes.
- Cuando haces una copia local del código, estás en tu propia **rama/branch** local que no tiene por qué afectar nunca al proyecto, pero a la vez es fácil integrar tu copia local al proyecto si se quiere. Y podemos hacer las ramas que queramos: una para probar una cosa nueva, otra para arreglar o corregir algo, otra para añadir funcionalidad, otra para mejorar algo, etc. Si lo hago mal en alguna de esas ramas, eso no afectará al proyecto porque esa rama se descartará y nunca se integrará, y el proyecto sigue su curso sin ella. Esto (**ramas/branching**) está integrado en Git para todos los usuarios que colaboran en un proyecto. Si la rama es buena, la gente la tomará. Si no es buena o no llevan a ningún lado, no afectará en nada al proyecto.

Conceptos clave:

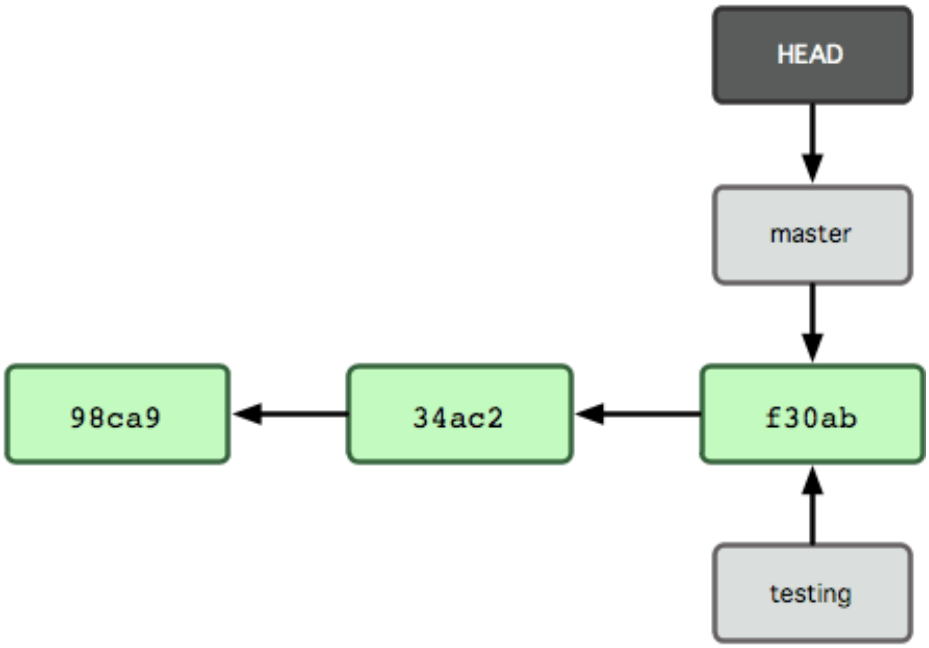
- Cuando hacemos un **commit** (confirmación) en Git, Git guarda un **"commit object"** que contiene un puntero a un **snapshot** (instantánea) de:
  - el contenido que hemos puesto en el **staging** (un concepto importante que veremos después)
  - el autor
  - un mensaje de documentación de ese commit
  - un puntero al commit anterior (o varios punteros si se trata de un *merge* ya que ese *merge* será de varios estados anteriores)



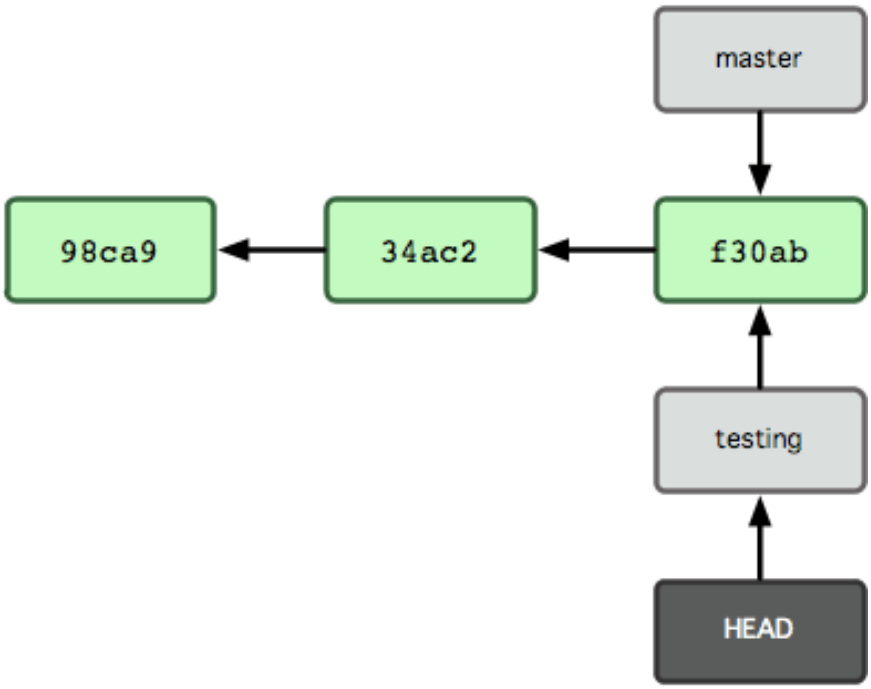

- Una rama es un puntero a un commit. La rama por defecto se llama **“master”** y es la rama oficial y de referencia que cuando se termine contendrá la versión en producción del proyecto. El puntero de la rama master se mueve automáticamente al último commit que vamos haciendo. Si creamos una nueva rama, se crea un nuevo puntero en el commit en el que estemos en ese momento ($git branch testing #crea una rama de nombre “testing” como en la siguiente figura).



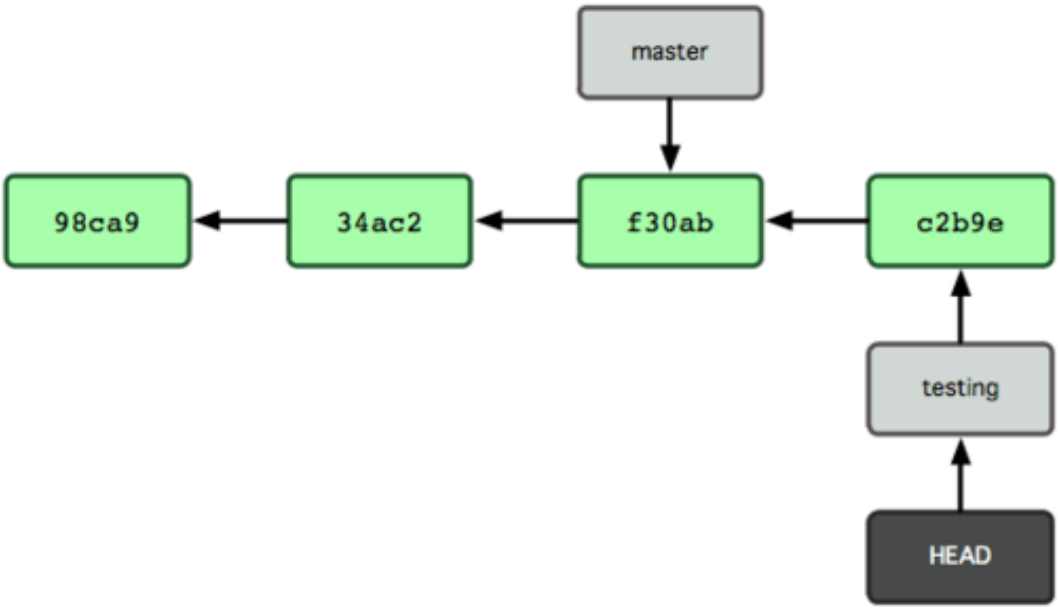

- Git sabe en la rama que estamos gracias al puntero **HEAD**. Con “git branch” se crea una rama pero no nos cambiamos a ella. Para eso se usa “git checkout”. Estábamos así:



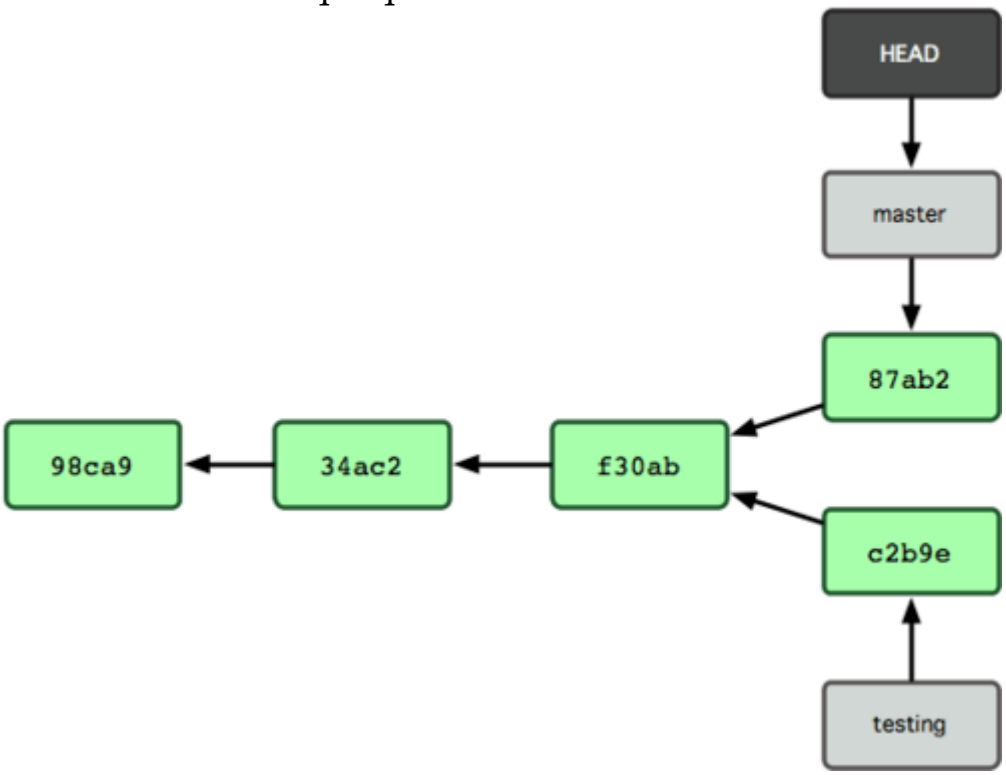

- Y ahora con $git chechout testing:




- Ahora estamos en la  rama “testing”. Si hacemos commits ahí, desarrollamos esa rama como se ilustra en la siguiente figura.




- Si nos vamos a la rama master ($git checkout master) y hacemos commits ahí, desarrollamos esa rama como se ve en la siguiente figura. Posteriormente las podremos mezclar o lo que queramos.




- El desarrollador trabaja en el "working tree" o "**working directory**" que va cambiando dependiendo de la rama que estemos. El desarrollador hace sus cambios en su working directory y luego decide si hace commit de ellos, o los integra con otras ramas, etc.

Esta es una breve descripción inicial de algunas características de Git. A continuación nos pondremos manos a la obra para seguir avanzando. Vamos a realizar dos sesiones de ejercicio. La primera trabajando únicamente en modo local y la segunda usando repositorios remotos.

**EJERCICIOS:**

En esta práctica vamos a hacer la primera toma de contacto con Git. Sobre todo trabajaremos en un repositorio local independiente que cada alumno se creará en su cuenta siguiendo las instrucciones del profesor.
Pasaremos a hacer las operaciones básicas en dicho repositorio: crearemos nuevos ficheros, haremos un commit de inicio de proyecto, seguiremos el desarrollo con nuevos commits. Haremos una rama/branch con algunas modificaciones y haremos merge con y sin conflictos.
Como parte del ejercicio, busca en la documentación de Git los comandos necesarios para cada ejercicio.

**1.-** Explica brevemente qué es un sistema de control de versiones distribuido y sus diferencias con respecto a uno centralizado. Entra en la web de Git y revisa la documentación. Busca manuales, tutoriales, etc. de Git. Describe y compara con Git algún otro sistema de control de versiones distribuido (Mercurial, etc.).

**2.-** Establece tu nombre de usuario y dirección de correo electrónico. Esto es importante porque las confirmaciones de cambios (*commits*) en Git usan esta información, y es introducida de manera inmutable en los commits que envías. Utiliza tu usuario y correo de la uco, por ejemplo:

```
$ git config --global user.name "i22lojal"
$ git config --global user.email "i22lojal@uco.es"
```

**3.-** Vamos a crear un nuevo repositorio desde cero (puedes usar uno tuyo o seguir el breve ejemplo que se describe a continuación). Crea en tu cuenta un nuevo directorio que se llame “reposlycs”. Entra en dicho directorio e inicializa el repositorio (`$git init`). Comprueba la creación del directorio .git. ¿Qué función tiene este directorio?

**4.-** Crea un directorio que cuelgue del anterior llamado “include”.

**5.-** En el directorio “reposlycs” escribe un fichero llamado “helloworld.c” que contenga el siguiente código:

```
#include "./include/myinclude.h"

int main(){
            f();
     return 0;
}
```

y un fichero “include/myinclude.h” que contenga:

```
#include <stdio.h>
void f()
{
printf(“Hello World \n”);
```

```
}
```

**6.-** Añade estos ficheros a tu repositorio y comprueba el estado del repositorio con “git status”.  Añade los cambios al stage (`$git add ...`). Haz el primer commit con el mensaje “Initial commit” y vuelve a comprobar el estado de tu repositorio.

**7.-** Compila y comprueba lo que ocurre con el ejecutable generado. ¿Se podría añadir el ejecutable al staging y luego al commit?. Haz una prueba. ¿Tendría sentido hacer commit de un ejecutable?.

**8.-** Abre una rama que se llama “branchA” y modifica el fichero de la función f() con lo necesario para que f() quede así:

```
void f()
{
     char c1[100]= "Hello world";
     char c2[100]= “, I am <your name here>”;
     printf("%s\n", strcat(c1, c2));
     return 0;
}
```

**9.-** Realiza el commit con los cambios mencionados en la branchA y el mensaje “Now with strcat()”.

**10.-** Cambia a la rama master (`$git checkout master`). En la rama master añade un comentario al inicio del fichero “helloworld.c” con la fecha y tu nombre como autor del fichero. Realiza el commit de este cambio en la rama master. Con el mensaje “Now with date and author”

**11.-** Desde la rama master, fusiona la rama master con la rama branchA. En este caso no hay ningún conflicto, ambas ramas han modificado distintos archivos. Es un merge sin conflicto.

**12.-** Repite el proceso del punto anterior pero ahora creando un conflicto (merge con conflicto).

**13.-** Sube este proyecto a GitHub.

**14.-** Bájate un proyecto de GitHub de uno de tus compañeros (o de cualquier otra persona) a tu cuenta local.