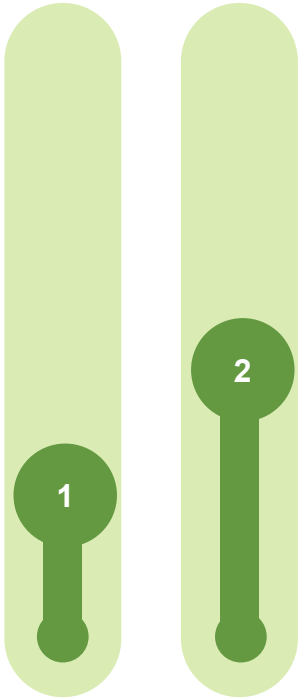
Principios Básicos de Versionamiento



¿Qué es un Sistema de Control de Versiones?

Es un software que controla y organiza las distintas revisiones que se realicen sobre uno o varios documentos.

Pero, ¿qué es una revisión? Se podría decir que una revisión es un cambio realizado sobre un documento, por ejemplo añadir un párrafo, borrar un fragmento o algo similar.

Supongamos que cargamos en un Sistema de Control de Versiones el siguiente código fuente:

```
lista=[25,12,15,66,12.5]
vector=np.array(lista)
print(vector)
```

Se añade como la revisión 1 del archivo. Una vez añadido, vemos que no compila, ya que nos falta importar la librería numpy y asignarle el sobrenombre np.

```
import numpy as np

lista=[25,12,15,66,12.5]
vector=np.array(lista)
print(vector)
```

Se vuelve a añadir ahora como la revisión número 2. De esta forma, se guarda el historial de las distintas modificaciones sobre un archivo, por lo que en cualquier momento podemos restaurar la revisión que queramos de un fichero.

¿Que pasa si no se cuenta con un sistema de Control de Versiones?

En el supuesto de un equipo que cuenta con 5 desarrolladores, que se encuentran realizando un proyecto. Puede ser que el proyecto esté muy claro qué partes tiene que hacer cada uno, pero esto no suele ser así.

Estos desarrolladores van subiendo a un servidor FTP las distintas modificaciones que realizan sobre el código. Uno sube el código fuente ***clase_A.py***, otro ***clase_B.py***, etc... Sin embargo, llegará un momento en el que dos desarrolladores modifiquen el mismo archivo. Veamos un ejemplo claro, con el siguiente código:

A continuación se encuentra el archivo de referencia, sobre el cual se harán ciertos cambios que provienen de distintos desarrolladores.

```
class Humano():
    def __init__(self, edad, nombre, ocupacion):
        self.edad = edad
        self.nombre = nombre
        self.ocupacion = ocupacion

persona = Humano(30, "Estuardo", "Ingeniero")
```

Ahora, el ***desarrollador 1*** decide agregar un método para retornar el valor de cada argumento del objeto persona porque ha notado que el objeto se ha creado pero realmente NO HACE NADA!

```
class Humano():
   def __init__(self, edad, nombre, ocupacion):
      self.edad = edad
      self.nombre = nombre
      self.ocupacion = ocupacion

def presentar(self):
      presentacion = ("Hola soy {}, mi edad es {} y mi ocupación es {}")
      print(presentacion.format(self.nombre, self.edad, self.ocupacion))

persona = Humano(30, "Estuardo", "Ingeniero")
persona.presentar()
```

Realiza el cambio y sube el archivo al servidor FTP, eliminando el que ya estaba ahí. Ahora resulta que en el intervalo de tiempo en el que el ***desarrollador 1*** estaba realizando la función **presentar(),** el ***desarrollador 2*** decide realizar otra función.

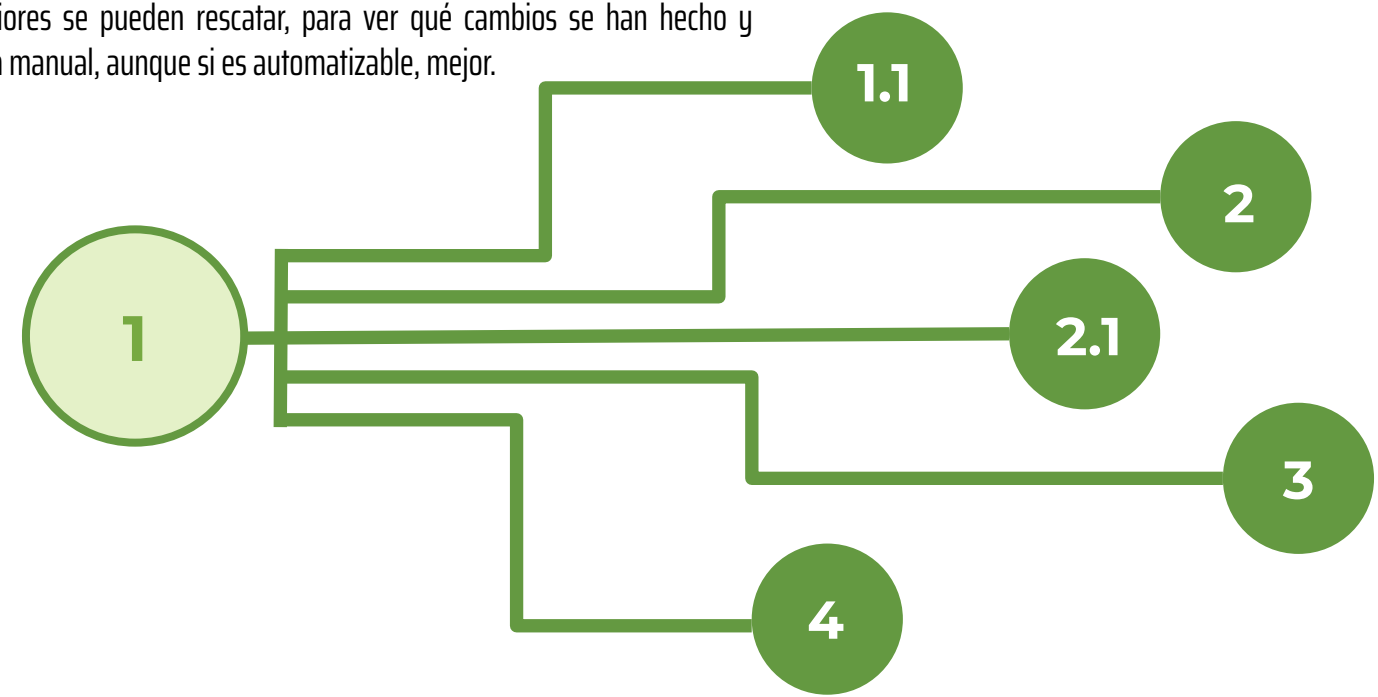
Desarrollador 2 decide crear una función para modificar el atributo ocupación.

```
class Humano():
   def __init__(self, edad, nombre, ocupacion):
      self.edad = edad
      self.nombre = nombre
      self.ocupacion = ocupacion

def contratar(self, puesto):
         self.puesto = puesto
         print ("{} ha sido contratado como {}".format(self.nombre, self.puesto))
         self.ocupacion = puesto

persona = Humano(30, "Estuardo", "Ingeniero")
persona.contratar("Project Manager")
```

El ***Desarrollador 2*** descargó la revisión anterior a la modificación de ***Desarrollador 1***, por lo que sube el archivo y elimina la versión anterior, que es la que había hecho ***Desarrollador 1***, eliminando su trabajo. Por tanto la próxima vez que ***Desarrollador 1*** compruebe el servidor, verá que lo que hizo ya no está.

Aquí entran en juego los sistemas de control de versiones. Si el grupo de desarrollo usará uno, esto no pasaría, ya que estos sistemas vigilan sobre qué líneas se han hecho estas modificaciones, de forma que "verían" que los dos cambios no son incompatibles, y la nueva versión combinará (realizará un merge) las dos modificaciones.

Incluso sin esta función, el problema de conflicto, sería más fácilmente resoluble, ya que al mantener versiones anteriores se pueden rescatar, para ver qué cambios se han hecho y realizar esa unión de forma manual, aunque si es automatizable, mejor.



Términos básicos

01

Repositorio

Una base de datos en la que los cambios son almacenados.

02

Branch (rama)

Un proyecto puede ser bifurcado en un momento de tiempo de forma que, desde ese momento en adelante, dos copias de estos archivos puedan ser desarrollados a diferentes velocidades.

03

Checkout

El proceso de obtener una copia del proyecto desde el repositorio.

04

Commit

Una copia de los cambios hechos a una copia local es escrita o integrada sobre repositorio.

05

Fusión (Merge)

Une dos conjuntos de cambios sobre un archivo.