Hemos planteado las siguientes ideas:

1. Utilizaremos Github, el cual nos permitirá trabajar simultáneamente en el código.
2. Las reuniones las haremos presencialmente y vía online por medio de programas como AnyDesk (para compartir pantalla y poder controlar lo que se hace) y llamadas por whatsapp
3. Se utilizarán estructuras de datos como listas enlazadas y grafos. También implementaremos JSON para el manejo de entrada y salida
4. Utilizaremos librerías que nos ayudarán en el desarrollo del proyecto como matplotlib y networkX
5. Cada uno de los comandos se deberán correr sobre la consola de Ubuntu, dentro de la ejecución del programa de manera perpetua
6. Para que el programa sea perpetuo, es decir, no finalice hasta que el usuario indique el cierre del mismo, el programa se correrá en un ciclo infinito.
7. El programa recibirá, por parte del usuario, los parámetros pertinentes para darle flujo al programa, dichos parámetros se recibirán en primera instancia como una cadena, la cual será convertirá en un arreglo cuyos índices serán dichos parámetros.
8. Con el arreglo creado de la cadena que introduzca el usuario, se validarán los comandos, verificando la cantidad de parámetros que ha introducido, ejemplo, si el índice 0 del arreglo es “link” y el 1 es “/home” el comando será válido, sin se detecta mes índices, será un comando inválido.
9. Los comandos válidos que procesará el programa son:
    a. ls
    b. ls -1
    c. mkdir [NOMBRE DEL DIRECTORIO]
    d. help
    e. plot
    f. ln [RUTA DEL ARCHIVO]
    g. touch [NOMBRE DEL ARCHIVO]
    h. rm [NOMBRE DEL ARCHIVO]
    i. rmdir [NOMBRE DEL DIRECTORIO]
    j. cd [NOMBRE DEL DIRECTORIO]
    k. cd ..
    l. findbe [EXTENSIÓN DEL ARCHIVO]
    m. recover [RECUPERAR LO BORRADO]
10. Se creará un sistema que simule un gestro de archivos, donde se podrá generar archivos, directorios, subdirectorios, utilizando JSON.
11. La estructura JSON se guardará en un archivo, así el programa podrá leerlo en todo momento.

12. Al iniciar el programa se debe cargar el archivo con la estructura JSON, luego, deberá leer su contenido para crear un grafo con dicha información, se debe hacer el proceso inverso para guardar la información actual en el archivo.
13. El programa al iniciarse deberá mostrar un mensaje como cabecera, el cual contendrá:
    a. El nombre del programa.
    b. Programadores del proyecto.
    c. La versión del mismo.
    d. Descripción breve del programa.
14. El sistema deberá realizar las cuatro operaciones básicas:
    a. Crear archivos, carpetas y enlaces a archivos.
    b. Leer rutas, grafos, archivo en disco duro, registros de archivos y carpetas.
    c. Actualizar ruta, archivo y carpeta.
    d. Eliminar ruta, archivo y carpeta.
15. Todo procedimiento debe trabajarse sobre el grafo.
16. Se tomarán fotografías y se creará un video que se proveerán como evidencia del trabajo realizado. Todo ello ira en archivos pdfs individuales, para luego ser enviadas para su posterior revisión.
17. Se utilizarán TDAs para manejar los procedimientos internos del programa.
18. Se implementará el uso de captura de fecha y hora para registrar las acciones del programa.
19. Se pondrá en practica las clases ya conocidas y programadas en la asignatura para generar listas enlazadas (LinkedList.py), grafos (Graph.py), nodos (Node.py), papelera de reciclaje (Trash.py)
20. Solamente se creará una ventana (interfaz gráfica) que se usará para embeber la imagen del grafo diseccionado en la pantalla del usuario, cuando este ejecute el comando “plot”.
21. El programa se codificará en idioma inglés con comentarios y salidas de pantalla en español.
22. El programa debe ser capaz de procesar caracteres latinos.
23. Implementaremos un alfabeto como guía para balancear (ordenar) el grafo, se comparara cada uno de los nombres de los enlaces, archivos y carpetas para ordenarlos alfabéticamente; evitando usar los operadores lógicos usuales “<” y “>” para las cadenas.