==========
A)  Video 1  https://youtu.be/ANF1X42_ae4
==========

1. Git salva instantáneas de nuestros ficheros (código + descripción)
      Instantánea-0
      Instantánea-1
      ....
      Instantánea-n
   Se puede volver a una instantánea anterior

2. Una aplicación está formada por muchos archivos
   (por ejemplo, si trabajamos con un FrameWork)
   Necesitamos una instantánea fija de muchos archivos
   (Git toma una instantánea de todos los archivos)

3. Git permite crear ramas
   Un programa o aplicación se puede dividir en varias ramas o flujos de trabajo.
   En una aplicación pueden trabajar varios programadores.
   Así cada programador se centra en uno de los flujos de trabajo o ramas.
   Cada programador hace instantáneas de sus ramas.

4. Git después puede unir o fusionar las diferentes ramas de una aplicación en una única rama.

5. Git, al fusionar las diferentes ramas, puede detectar conflictos
   (ejemplo: dos programadores hacen modificaciones en una misma parte de la aplicación. Git detecta este conflicto y lo intenta solucionar)

6. Git aumenta su potencia al unirse con GitHub.
   Se descarga Git y se instala. Lo importante es la Consola de Git.
   Visual Studio Code es un IDE que está preparado para trabajar con Git.
   Pildoras Informáticas lugar donde están estos vídeos sobre Git.

```
========================================
B)  Video 2  https://youtu.be/qk-GWtcdQek
========================================

1. Se verán estos aspectos:
      Crear repositorio
      Agregar al repositorio
      Respaldar en el repositorio
2. Tenemos tres zonas:
      - Directorio de trabajo
        (carpeta donde están nuestros ficheros)
      - Área de ensayo
        (Staging area)
      - Repositorio local
        (lugar donde se guardan las instantáneas)
3. Una primera aproximación a los comandos y su significado
      $ git init
        Se ejecuta sólo una vez, al comienzo, y así se crea el repositorio
        local, en donde vamos a hacer un seguimiento de nuestros ficheros.
      $ git add
         Con esta orden se llevan los archivos del directorio de trabajo
         al área de ensayo
      $git commit
         Con esta orden se llevan los archivos del área de ensayo
         al repositorio local
4. Creamos una carpeta donde pondremos nuestros ficheros y será
   el Directorio de trabajo.
   Ahora, en esta carpeta, botón derecho escoger "Git Bash Here" y se abre
   la consola de Git en esta carpeta.
   Escribimos "git init" y se crea el Repositorio local en esta carpeta.
   (viendo ocultos aparece .git)
5. Indicar lo que va a ser seguido por Git
      $ git status -s
      (da el listado de todo lo que hay en el directorio de trabajo;
       los ficheros, con ?? delante, es que no se están siguiendo)
   Si queremos seguir sólo un fichero:
      $ git add <fichero>  (git add caso.txt)
      $ git status -s
      (muesta el archivo, con A delante; es que está agregado al área de ensayo,
       se le está haciendo un seguimiento)
6. Para hacer una instantánea
      $ git commit -m "<descripción>"
   Si ahora hacemos
      $ git status -s
      (No muestra lo que está respaldado, lo que está en el repositorio.
       Sí muestra lo que se está siguiendo, lo que está en el área de ensayo)
   Si ahora hacemos una modificación en un fichero
      Hasta que no hagamos "git add <fichero>", no se agregará
      al área de ensayo
7. Para listar todas las instantáneas que están en el repositorio local
      $ git log --oneline
      (listará tantas instantáneas como commit se hayan realizado)

8. Para restaurar una instantánea concreta
      $ git reset --hard <código de la instantánea>
   Si ahora hacemos
      $ git log --oneline
      (muestra las instantáneas actuales. Si se restaura a
       una instantánea concreta, las instantáneas posteriores se pierden)

----------------------------------------
git config --global user.username "<nombre>"
git config --global user.email "<email>"
```

==========
C)  Video 3  https://youtu.be/0UlqvTJzOL4
==========

1. Veremos
   - cómo subir un repositorio local a GitHub y
   - cómo hacer un seguimiento a todos los ficheros
   $ git add .
     (agrega al área de ensayo todos los ficheros)
   $ git log --oneline
     (veremos todos los commit)

2. Cómo hacer dos comandos en uno
   $ git add .
   $ git commit -m "<descripción>"
     (estos dos comandos se pueden hacer en uno, ver a continuación:)
   $ git commit -am  "<descripción>"

3. Para corregir una descripción de un commit
   - abrir editor vim de esta forma:
   $ git commit --amend

4. Cómo subir a GitHub
   - Primero registrarse
   - En el segundo bloque (or push ..)
       1ª línea es para subir el repositorio local a GitHub
       3ª línea es para nuevas subidas a GitHub
   - Ahora se puede ver el repositorio subido
     (informa de los commit)
     En un commit, en el botón <>, se ve el estado, en esa instantánea,
     de los fichero
   - En GitHub podemos ver el estado de los ficheros, en los diferentes
     commit (aparece la descripción de los commit)

==========
D)  Video 4  https://youtu.be/axXlYdyDD3I
==========

1. Veremos
   - cómo editar archivos desde GitHub (en remoto) y después
     trasladarlos a local, sincronizarlos
   - cómo crear tags y para qué sirven
   - cómo clonar un repositorio, que está en remoto,
     a local (traerlo a local)

2. Modificamos en remoto un archivo
   - lo editamos para realizar los cambios que queramos
     y pulsamos el botón "commit changes"
     (en este momento se guardan los cambio en remoto)

3. Cómo traer lo de remoto a local
   - Recordar que para subir de local a remoto se utilizaba:
   $ git push
   - Pues para bajar de remoto a local se utiliza:
   $ git pull
   - git fech es muy parecido a git pull (pero se verá más adelante)

4. Ahora queremos guardar una primera versión de nuestro proyecto (repositorio)
   (funciona bien, nos gusta como está)
   - Cómo hacemos esto: con tags podemos especificar versiones
     de nuestro proyecto
   - En GitHub, si vamos a nuestro proyecto, tenemos 0 tags
   - En local, en la consola, hacemos
   $ git log --oneline (y no vemos ningún tag)
     Ahora escribimos esto:
   $ git tag 11-02-21v1 -m "Versión 1 del proyecto"
     Si ahora hacemos
   $ git log --oneline (ya nos indica que creamos una tag)
     Para subir este tag ponemos
   $ git push --tags
   - Ahora en GitHub, actualizando, ya está la tag y nos ofrece la opción
     de descargar esta versión en un zip
     En el futuro, cuando se creen más tags, tendremos las diferentes
     versiones de nuestro proyecto.

5. Ahora simulamos un desastre
   (eliminamos en local la carpeta de nuestro proyecto)
   - cómo recuperar nuestro proyecto?
   - vamos a GitHub y en el botón Code vemos que podemos clonar
     el repositorio (nos muestra la URL)
   - En la consola, en local, nos ponemos en la carpeta donde
     queremos clonar el proyecto y escribimos
   $ git clone <URL>

```
========================================
E)  Video 5  https://youtu.be/q9LJIHDgJtE
========================================

1. Rama o branch (es una línea de tiempo)
      - A esta línea de tiempo se le llama master
      - Se pueden crear vias líneas de tiempo paralelas
      - Así tenemos la línea de tiempo master
        (se puede volver a commit anteriores en la rama master)
      - Se puede añadir una línea de tiempo adicional para
        hacer pruebas y ver cómo quedaría el proyecto
      - En proyectos en los que hay varios programadores
        se pueden crear varias líneas de tiempo.
        (en una de esas ramas se trabajaría sobre unos ficheros
         y en otra se trabajaría sobre otros ficheros)
      - Se puede crear una rama adicional. Así estaría la master
        y además la rama adicional. Y se puede trabajar a la vez
        en las dos ramas.
        (esta rama adicional sería, por ejemplo, para pruebas)

2. Fusión de ramas (merge)
      - Se puede trabajar con muchas ramas a la vez.
      - Depués se pueden unir las ramas adicionales en la
        rama master, es una fusión.
      - Advertencia: se puede modificar el mismo fichero en dos ramas
        diferentes
        (aquí surge un conflicto: se está modificando lo mismo en
         el mismo fichero en dos ramas diferentes).

3. Caso práctico
      - Pasos: a)creamos una rama adicional, b)se harán cambios en las dos
        ramas y después c)se fusionarán.
      $ git log --oneline (se ven los commit)
      - En el último commit aparecee la palabra master.
        (estamos en la rama master, no hay más ramas)
      - Supongamos que queremos modificar ciertos archivos
        para ver cómo quedaría el proyecto o bien deseamos
        hacer pruebas.
        (en este caso lo que se debería hacer es crear
         una nueva rama adicional, para realizar los cambios
         que deseamos)

4.- Crear rama adicional
      $ git branch <nombre> (ej: git branch adicional)
      - Hacemos un "git log --oneline" y aparecerán 2 ramas
      $ git branch
        (nos muestra las ramas existentes y en que rama estamos)
      - Las modificaciones que hagamos serán en la rama donde
        estemos.
      - Para movernos a una rama:
      $ git checkout <nombre-rama>
        (ej: git checkout adicional)
      $ git branch
        (ahora nos indica en la rama donde estamos)
        Hacemos cambios y ya son en la nueva rama donde
        estamos (por ejemplo, en la rama adicional)
```

- Seguidamente agregamos las modificaciones al
  área de ensayo y hacemos instantánea:
$ git add .
$ git commit -m "<descripción>"
$ git log --oneline
  (se ve el commit en la nueva rama)
- Al movernos a la rama master, todo lo hecho en la
  otra rama (la rama adicional) no estará.
$ git checkout master
  (el proyecto vuelve a estar como lo teníamos en la
   rama master)
- Si nos movemos de nuevo a la rama adicional,
  el proyecto estará como lo dejamos, cuando la
  abandonamos.
- Es una forma de trabajar con dos copias del mismo
  proyecto e ir de una a otra.
- Nos movemos de nuevo a la rama master y modificamos
  un fichero que ya habíamos modificado en la rama
  adicional.
- Seguidamente agregamos las modificaciones al
  área de ensayo y hacemos instantánea:
$ git add .
$ git commit -m "<descripción>"
$ git log --oneline
  (ahora no se ve el commit hecho en la rama
   adicional)
- Si nos movemos a la rama adicional, vemos
  el commit hecho en esta rama, pero no el hecho en
  la rama master.
  (es como si fueran dos proyectos diferentes)

5. ¿Cómo fusionamos lo hecho en ambas ramas?
- Para hace un merge hay que moverse obligatoriamente
  a la rama master.
  (el merge se debe hacer desde la rama master)
$ git merge <nombre-rama>
  (nos informa de que hay conflictos y que no
   se hizo el merge)
- Veremos en otro vídeo cómo resolver este conflicto