**Ejercicio sobre control de versiones y Git**

Algunos enlaces que hemos comentado:

<https://git-scm.com/>

Repositorios de proyectos:

<https://github.com/>

<https://bitbucket.org/>

Software de Git gráficos:

<https://desktop.github.com/>

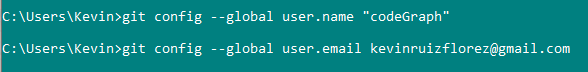
<https://www.sourcetreeapp.com/>

**Ejercicios propuestos:**

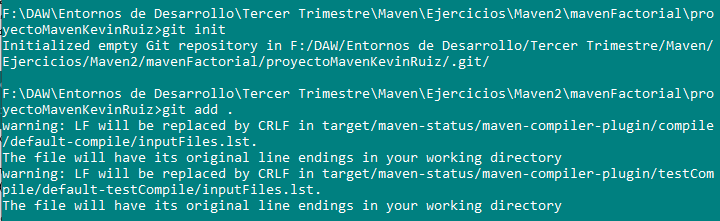
1. ¿Para qué se usa un Sistema de Control de Versiones (SCV)? (Razones por la que es necesario)

Se usa para poder realizar un seguimiento a todas las versiones que hemos realizado en nuestro programa con el fin de almacenar todos aquellos cambios que han sido realizados sobre los archivos de nuestro proyecto.

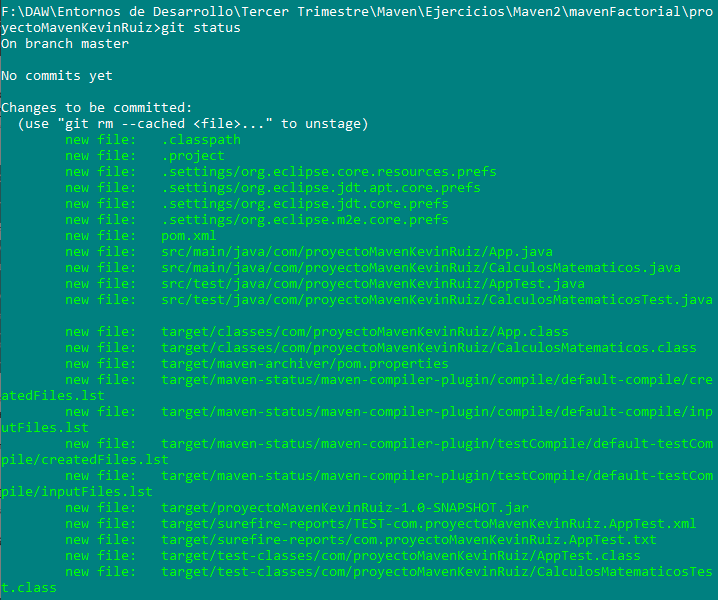
1. ¿Qué significan los siguientes conceptos en un SCV? Es un lenguaje habitual en un control de versiones. Podéis preguntarme sobre ellas en la clase si no entendéis algún concepto.
   1. **Repositorio** Son directorios en los cuales se encuentran todos los archivos de nuestro proyecto, junto con sus versiones y distribución de estos en la estructura de ramas que hayamos establecido.
   2. **Versión** Es una captura del desarrollo de un código, el estado en el que se encontraba nuestro archivo en el momento de realizar la versión. Las versiones surgen cuando un archivo se añade, se modifica o se remueve, es decir, cuando el estado global de nuestro proyecto se altera.
   3. **Commit (confirmación o check-in)** Es un cambio que hemos realizado en alguno(s) de los ficheros de nuestro proyecto el cual se guarda en el historial de Git con tal de poder ver los cambios que se le han ido realizando al proyecto.
   4. **Master (o trunk, tronco)** Es la rama principal de todo proyecto, de la cual se derivan primeramente todas las ramas de nuestro proyecto. Es normalmente la rama en la cual se encuentran las versiones estables de nuestro proyecto.
   5. **Branch (rama)** Son divisiones que realizamos en nuestro proyecto para poder individualizar los cambios que se le realicen al mismo según distintos criterios, para que en dicho caso se presente algún error en un commit de la rama máster hacer el proceso del debug más simple.
   6. **Check-out (desplegar)** Se usa para poder moverse entre las distintas ramas de nuestro proyecto.
   7. **Origin (Repositorio origen y copia local)** Se le denomina al repositorio original del cual se realiza una copia en local.
   8. **Merge (fusión)** Se le denomina a la función de juntar los cambios realizados en un commit de una rama junto con el último commit de la otra rama con la que queremos juntar.
   9. **Conflicto** Suele ocurrir un conflicto a la hora de realizar un merge cuando los archivos de ambas ramas han sido modificados en las mismas partes de los archivos, por lo cual el asistente no sabe cuál versión permanecer.
   10. **Push** Sirve para sincronizar los cambios hechos en local con el repositorio remoto.
   11. **Pull** Es una solicitud que se realiza una vez hechos los cambios en un proyecto que se encuentra en una rama hija con el fin de fusionar dichos cambios con la rama principal. Dicha solicitud se suele supervisar por un encargado del repositorio.
   12. **Fetch** Se usa para descargar las ramas de un repositorio remoto al local evitando copiarlo todo.
   13. **Stage área** Es el área en el cual se colocan todos los archivos de los cuales queremos realizar un nuevo commit en la rama que nos encontramos de nuestro proyecto.
2. Hacer las siguientes acciones con Git (Algunas vistas en el primer vídeo).
3. Instalar git
4. Configurar vuestro usuario en Git



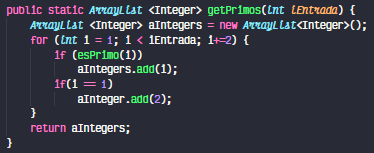
1. Añadir un proyecto de Eclipse para controlar sus versiones con Git. Utilizar el creado con Maven.



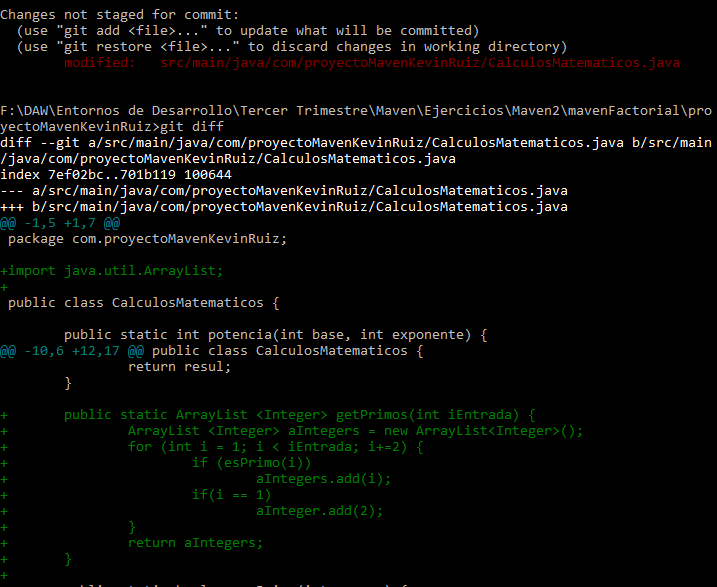
1. Comprobar con git status que se ha creado correctamente.



1. Después de añadir el proyecto a Git: Modificad la clase CalculosMatematicos.java en el proyecto. Por ejemplo, añadid un método a la clase que devuelva una lista (o un array) con todos los números primos hasta un número dado como argumento.

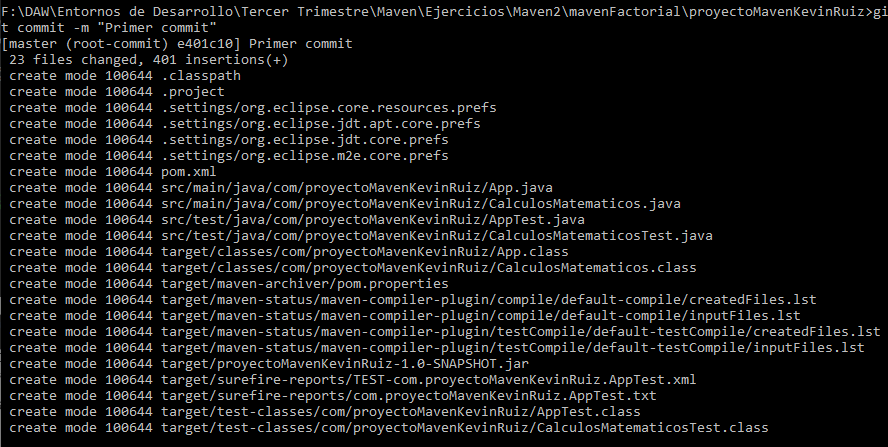


1. Comprobad con git status cuál es el estado de los ficheros ahora, después de añadir esa clase. Probad git diff

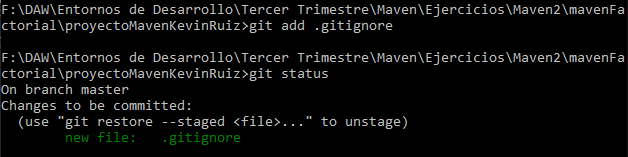


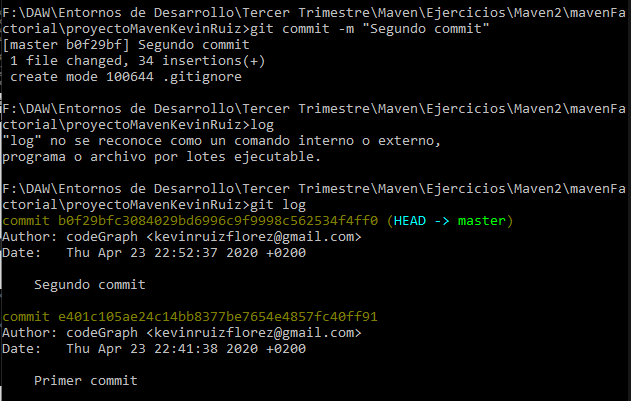
1. Poner las modificaciones preparadas para commit y hacer commit

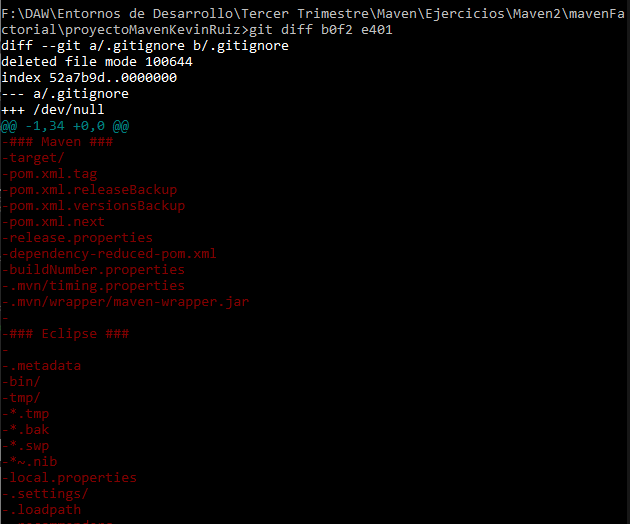




1. Añadir un fichero .gitignore para eliminar ficheros del “tracking” de Git. Buscad en internet un ejemplo de .gitignore para java+Maven+eclipse. Añadir el .gitignore a “preparado para commit” y hacer otro commit. Volved a usar git log, git show, git diff, etc.







1. Daros de alta en Github y subir el proyecto a Github, compartirlo de forma pública.

<https://github.com/codeGraphK/mavenFactorial>