

**Desarrollo e Integración del Software**

*PRC 01: Gestión de versiones con GIT y GitHub*

Verónica Ciancia Alonso | Guzmán Oñate | David Nieto Peña

Escuela Politécnica Superior

Grado en Ingeniería Informática

13 de marzo del 2020

Curso 2019 – 2020

Tabla de contenido

[1. Objetivo de la práctica: 3](#_Toc35019387)

[1.1 Contenido del documento 3](#_Toc35019388)

[2. Introducción 4](#_Toc35019389)

[2.1 Git y GitHub 4](#_Toc35019390)

[3. Desarrollo 5](#_Toc35019391)

[3.1 Errores producidos durante la práctica 6](#_Toc35019392)

[4. Conclusión 7](#_Toc35019393)

[5. Referencias 8](#_Toc35019394)

# Objetivo de la práctica:

El objetivo de la práctica 1 es llevar a cabo los conocimientos adquiridos a lo largo de las clases sobre Git y el alojamiento de repositorios GitHub.

## Contenido del documento

El documento contendrá todos los puntos indicados en el índice anteriormente presentado, y serán comentados en sus correspondientes apartados. También se incluirán tablas, imágenes o esquemas en caso de que fuera necesario.

Por último, se contempla la posibilidad de consultar los anexos y bibliografía en caso de que hubiera la necesidad de realizar dicha tarea, estos se encuentran al final del documento.

# Introducción

Los sistemas de control de versiones son programas que permiten almacenar los cambios que se van produciendo en diferentes ficheros durante su creación. Dichos sistemas permiten el desarrollo colaborativo, conocer el estado actual del proyecto y llevar un registro de quién y cuándo, se realizan dichos cambios. (Desarrolloweb.com, s.f.)

## Git y GitHub

*Git* es un sistema de control de versiones distribuido y se utiliza fundamentalmente para llevar a cabo un histórico de los cambios que se van produciendo en un proyecto de desarrollo, pero se podría implementar a cualquier tipo de archivo. (Control de Versiones)

Una de las principales características de *Git* es que cada usuario va a disponer de una copia local del historial completo del repositorio, a lo que se denomina Gestión Distribuida.

El almacenamiento de la información en Git va a tener tres estados:

* Confirmado (*Commited*): Significa que los datos están almacenados de manera segura en la base de datos local.
* Modificado (*Modified*): Significa que se ha modificado el archivo pero todavía no sea confirmado en nuestra base de datos.
* Preparado (*Staged*): Significa que se ha marcado un archivo modificado en su versión actual para que vaya a la próxima confirmación.

*GitHub* es uno de los alojamientos de repositorios de software más importantes que se pueden utilizar con Git hoy en día.

(Deustoformacion.com, s.f.)

# Desarrollo

Uno de los participantes del grupo ha creado un repositorio (***Practica1DISExtraordianrio***) con su cuenta de GitHub (***Davixin***) y nos ha invitado a participar en el repositorio como colaboradores.

A continuación se debería haber creado la estructura de lo que iba a ser el proyecto de java, con el *main* y sus clases en la rama *master* pero en este caso al tener un proyecto base (la entrega en ordinaria) hemos decidido comenzar desde ese punto. Posteriormente hemos creado la rama *develop* donde se irán guardando las partes de código terminadas de cada componente del grupo.

Cada integrante del grupo va a tener su propia rama procedente de *develop* en las que cada uno tendrá el código sin terminar o que esté en proceso de creación. Los nombres de las ramas serán:

* *developGuzman:* Perteneciente al usuario ***Guzonate***
* *developDavid:* Perteneciente al usuario ***Davixin94***
* *developVeronica:* Perteneciente al usuario ***VeronicaCian***

Se llevó a cabo la división del trabajo para poder realizarlo de forma equitativa y finalmente decidimos no trabajar en paralelo para así evitarnos la posibilidad de que se produzcan conflictos entre nuestros códigos.

La rama *master* no la hemos utilizado en ningún momento ya que decidimos que solo se utilizará para subir aquello que esté terminado y verificado su funcionamiento.

Cada vez que alguno de los integrantes quería trasladar su parte del código terminada realizaba lo siguiente:

* *git add .*
* *git commit -m”Mensaje del commit”*
* *git push* (Guardar los cambios en la rama correspondiente al usuario (*developUsuario*))
* *git checkout develop* (Nos cambiamos a la rama *develop*)
* *git merge developUsuario* (Incorporamos en *develop* los cambios realizados en la rama *developUsuario*)
* *git push* (Guardar los cambios en la rama *develop*)

Cuando los demás querían descargarse los cambios que se han producido en *develop* y así poder seguir trabajando se realizaba lo siguiente:

* *git pull* (Estando en la rama *develop,* descargar en nuestro repositorio local los cambios realizados por otro usuario)
* *git checkout developUsuario* (Cambiamos de rama)
* *git merge develop* (Incorporamos los cambios producidos en develop a *developUsuario*).

Al comprobar que nuestro programa era capaz de registrar los datos introducidos por consola por el usuario en un archivo XML, se realizó el archivo DTD. Este documento define la estructura del documento XML, es decir, los elementos y atributos. Por tanto, si el DTD es válido, significará que el archivo XML también es correcto.

Una vez que nos hemos asegurado de que todo funciona correctamente hemos realizado lo siguiente:

* *git checkout master* (Estando en la rama *develop* los cambiamos a la rama *master*)
* *git merge develop* (Fusionar el proyecto terminado que hay en la rama *develop* a la rama *master*)
* *git push* (Guardar los cambios en el repositorio ***Practica1DIS***)

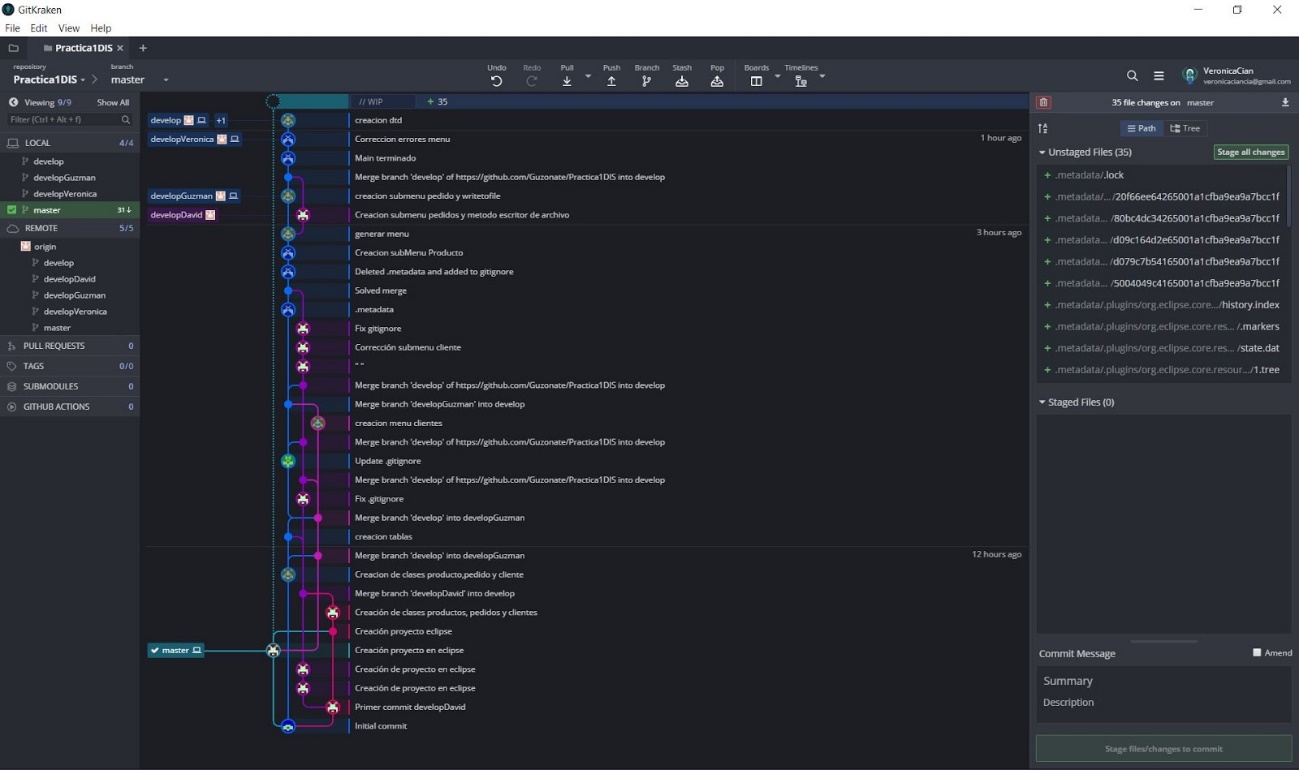


Ilustración : GitKraken - Repositorio Practica1DIS

Enlace repositorio: [***https://github.com/Guzonate/Practica1DIS***](https://github.com/Guzonate/Practica1DIS)

Enlace nuevo repositorio: [***https://github.com/davixin94/Practica1DISExtraordinario***](https://github.com/davixin94/Practica1DISExtraordinario)

## Errores producidos durante la práctica

Eclipse creo una carpeta de metadatos (.metadata) en la que almacenaba información acerca del proyecto creado en ficheros tanto de texto como binarios. Esto al realizar *merges* resultaba en conflictos informados desde la herramienta de consola.

Una vez resueltos estos conflictos se procedió a eliminar la carpeta del repositorio, para lo que se realizó un *commit* que no la contenía y una actualización del *.gitignore*.

De esta forma, al volver a crearse la carpeta por parte de Eclipse, esta no es supervisada por git.

## Errores Corregidos

En primer lugar hemos corregido la generación del xml ya que el formato que se generaba anteriormete no era el correcto.

El arraylist de los productos dentro de pedidos lo hemos puesto de tipo producto en vez de string

Hemos eliminado todos los problemas que nos direon con el metadata en la anterior convocatoria

# Conclusión

Una vez finalizado el trabajo, hemos podido observar que el desarrollo de un proyecto conjuntamente fue más dinámico gracias al control de versiones que se ha realizado con *Git* y *Git Hub*. Para conseguir un desarrollo más ordenado se han seguido reglas establecidas por *Git-Flow.*

Este tipo de proyecto ha permitido poder ver los beneficios de trabajar con esta dinámica ya que las cargas de trabajo se han podido repartir, pero al ser un proyecto de pequeñas dimensiones, no se ha podido comprobar el desarrollo con control de versiones. Es decir, si se hubiese desarrollado una herramienta funcional, como una herramienta web, podrían haber aparecido más conflictos.

También destaca el uso de *GitKraken*, ya que ha servido para visualizar gráficamente como evolucionaba el desarrollo del proyecto en las diferentes ramas.

# Referencias

*Control de Versiones.* (s.f.). Obtenido de https://moodleufv.ufv.es/pluginfile.php/896170/mod\_resource/content/0/%282019-2020%29%20tema1.%20version%20control.%20git.pdf

*Desarrolloweb.com*. (s.f.). Obtenido de https://desarrolloweb.com/articulos/introduccion-git-github.html

*Deustoformacion.com*. (s.f.). Obtenido de https://www.deustoformacion.com/blog/programacion-diseno-web/que-es-para-que-sirve-github