UNIVERSIDAD FRANCISCO DE VITORIA

ESCUELA POLITÉCNICA SUPERIOR

Desarrollo e integración del software

Práctica I:

Gestión de versiones con GIT y Github

Aaron Hoffman García

Alejandro Berlinches Bocanegra

Jaime Pérez Palomera

Mario Fernández Burgos

Marzo 2020

Índice:

Introducción:

El objetivo de esta primera práctica es demostrar los conocimientos adquiridos a la hora de crear un fichero XML con su respectivo DTD, además del manejo de la herramienta Git. A su vez, se usan metodologías de colaboración, en este caso Gitflow, combinado con una serie de modificaciones que consideramos convenientes para esta práctica.

Desarrollo:

Durante el desarrollo de esta primera práctica nos surgieron algunos problemas relacionados con cómo íbamos a usar finalmente git. Al comienzo del proyecto se creó un repositorio en una de las cuentas de *Github* en uno integrantes del grupo. Cuando comenzamos a desarrollar y a usar git se propuso hacer *Fork* del repositorio principal, de esta manera cada uno de nosotros teníamos una copia del repositorio en nuestras cuentas de Github. Esto se diferencia de la alternativa 'clone', en la que el para hacer cualquier cambio en el repositorio es necesario tener permisos.

Con el paso del tiempo nos dimos cuenta de que este método no era muy eficiente, aunque si viable, ya que al comienzo de esta práctica decidimos hacer *pull-request* cuando tuviésemos que aplicar algún cambio al repositorio. Cabe destacar que todos teníamos los mismos permisos sobre el repositorio principal. Por otro lado, se decidió usar *git-flow*, que no deja de ser un conjunto de extensiones para Git que nos permite gestionar de una manera mucho más eficiente las ramas en nuestro repositorio. La principal ventaja que tenemos con *git-flow* es que no íbamos a modificar las mismas porciones de código de algún integrante del proyecto. A su vez, proporciona una forma de organizarse bastante sencilla para un grupo pequeño. Estuvimos trabajando con una de las opciones que nos ofrece esta herramienta que es trabajar con tres ramas: *feature, develop y master.*

Las ramas feature, se utilizan para construir la funcionalidad de un programa, funcionalidad la cual irá posteriormente a develop. Una vez está ahí, cuando sea conveniente para realizar una 'release', se mergeará la rama develop con master.

Con esta ayuda nos aseguramos de que nosotros mismos no vamos a afectar el trabajo de otro miembro del equipo indirectamente. Todo lo que podemos hacer con *git-flow* también lo podríamos hacer con Git, pero para no tener una mala metodología como nos pasó al principio del proyecto, *git-flow* nos obliga a seguir un flujo apropiado de trabajo y nos ayuda a trabajar mejor con nuestros proyectos.

Para asegurar la calidad del código y evitar que hubiesen cambios innecesarios, utilizábamos una metodología que consistía en que cuando alguien había acabado una *feature*, tenía que realizar siempre una pull request hacia *develop*, la cual era aprobada solamente cuando dos personas o más la habían revisado. No siempre seguimos este mecanismo, pero nos ayudó en varias ocasiones. Ejemplo de ello fue cuando un miembro del grupo subió una rama con una gran cantidad de archivos que no pertenecían al proyecto. De no haber existido el método explicado previamente, todos esos ficheros habrían acabado en la rama *develop*, y habría complicado la limpieza del código. En cambio, con este mecanismo, logramos evitarlo, diciéndole al causante del problema que lo arreglase en su rama antes de que le pudiésemos aprobar la pull request.

Otro de los conflictos sucedidos fue un problema de comunicación en el que dos compañeros sin conocer exactamente lo que estaba haciendo el otro en ese momento hicieron una *pull request* de dos ramas que cambiaban el mismo fichero. Una de ellas se mergeó antes que la otra, por lo que cuando queríamos juntar la otra rama a *develop*, hubieron bastantes conflictos. Para solucionarlos, utilizamos una extensión del editor de código 'Visual Studio Code', que te permite gestionar conflictos de manera gráfica y bastante sencilla. El que tuvo los conflictos tuvo que juntar ambos cambios, no pudiendo descartar ninguno de ellos.

Para aumentar la eficiencia durante el desarrollo del proyecto decidimos dividirnos el trabajo entre los integrantes del grupo, para dos de los miembros del equipo se encargarían de hacer las clases secundarias y de esta manera optimizar el tiempo de creación, otro se encargaría de realizar el *Main* y la clase *Pedido*, y por último el cuarto se encargaría de realizar el validador de XML en base a un DTD. Una vez terminadas estas tareas quedara realizar la parte de la escritura del XML con la que uno se pone a cargo y el resto ayudamos con cualquier problema que pueda surgir.

XML y DTD

Para crear el XML y DTD, hemos seguido las instrucciones que aparecían en la práctica, colocando los campos que debían de aparecer en cada elemento. La práctica giraba en torno al elemento 'pedido', el cual utiliza un 'cliente', una 'dirección' y un 'producto'. Un ejemplo de XML generado por nuestra práctica sería el siguiente:

```
<pedidos>
  <pedido>
    <productos>
      <producto>
        <nombre>Alfombra fashion</nombre>
        <codigo>213</codigo>
        <descripcion>Alfombra del siglo XII</descripcion>
        <stock>12</stock>
        <pendientes>29</pendientes>
        <localizacion>
          <pasillo>10</pasillo>
          <estanteria>1</estanteria>
          <estante>2</estante>
        </localizacion>
      </producto>
      <producto>
        <nombre>TV 4K Samsung</nombre>
        <codigo>209</codigo>
        <descripcion>Television curva cara</descripcion>
        <stock>29</stock>
        <pendientes>10029</pendientes>
        <localizacion>
          <pasillo>11</pasillo>
          <estanteria>4</estanteria>
          <estante>2</estante>
        </localizacion>
      </producto>
    </productos>
    <cliente>
      <nombre>Jose</nombre>
      <apellido>Gil</apellido>
      <email>jose@gmail.com</email>
      <telefono>661929292</telefono>
      <direccion>
        <calle>Cerro del Espino</calle>
        <codigoPostal>28221</codigoPostal>
        <numero>9</numero>
        <poblacion>Madrid</poblacion>
        <pais>Espanya</pais>
      </direccion>
    </cliente>
    <direccion>
      <calle>Cerro del Espino</calle>
      <codigoPostal>28221</codigoPostal>
      <numero>9</numero>
      <poblacion>Madrid</poblacion>
      <pais>Espanya</pais>
    </direccion>
  </pedido>
</pedidos>
```

La estructura es por lo tanto la siguiente:

El elemento 'pedidos', tiene uno o varios elementos de tipo 'pedido', el cual tiene a su vez uno o varios 'productos', que tienen en su interior una localización. También tienen un cliente, con una dirección personal, y una 'dirección', que hace referencia a la dirección de entrega.

El DTD quedaría de la siguiente forma:

```
<!ELEMENT pedidos (pedido)*>
<!ELEMENT pedido (direccion,(cliente)*,productos)>
<!ELEMENT direccion (calle,numero,codigoPostal,poblacion,pais)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT codigoPostal (#PCDATA)>
<!ELEMENT poblacion (#PCDATA)>
<!ELEMENT pais (#PCDATA)>

<!ELEMENT cliente (nombre,apellido,email,telefono,direccion)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>

<!ELEMENT productos (producto)*>
<!ELEMENT producto (codigo, nombre,descripcion,stock,pendientes,localizacion)>
<!ATTLIST producto cantidad CDATA   #IMPLIED>
<!ELEMENT codigo (#PCDATA)>
<!ELEMENT descripcion (#PCDATA)>
<!ELEMENT stock (#PCDATA)>
<!ELEMENT pendientes (#PCDATA)>
<!ELEMENT localizacion (pasillo,estanteria,estante)>
<!ELEMENT pasillo (#PCDATA)>
<!ELEMENT estanteria (#PCDATA)>
<!ELEMENT estante (#PCDATA)>
```

Explicación metodología:

Lo primero que hicimos al comenzar este proyecto fue crear las claves SSH de Git, de esta manera podíamos conectarnos fácilmente a nuestros repositorios sin tener que ingresar una contraseña, de esta manera lo que hacemos es que nuestros equipos puedan ser identificados y reconocidos por Git y Github a la hora de trabar en un repositorio, sin necesidad de estar introduciendo constantemente la contraseña.

Como habíamos dicho anteriormente, habíamos decidido usar *Git-flow*. Para ello tuvimos que crear *features*, que son ramas de Git con una copia exacta del contenido de la rama *Develop* en la que se trabajará posteriormente. Lo primero que tenemos que hacer es indicarle un nombre/alias a la rama de *feature*, en este caso tomaremos el ejemplo con la creación de la clase *Producto:*

```
Mario@DESKTOP-OIJ9KBH MINGW64 ~/Desktop/universidad/año3/cuatrimestre 2/Desarrol
lo e integracion del software/PR1_DIS (develop)
$ git flow feature start Producto
Branches 'develop' and 'origin/develop' have diverged.
And local branch 'develop' is ahead of 'origin/develop'.
Switched to a new branch 'feature/Producto'

Summary of actions:
- A new branch 'feature/Producto' was created, based on 'develop'
- You are now on branch 'feature/Producto'

Now, start committing on your feature. When done, use:

     git flow feature finish Producto


Mario@DESKTOP-OIJ9KBH MINGW64 ~/Desktop/universidad/año3/cuatrimestre 2/Desarrol
lo e integracion del software/PR1_DIS (feature/Producto)
$
```

Una vez que tenemos creada nuestra rama ya podemos empezar a trabajar sobre esta, que en nuestro caso se llama *feature/Producto*. Cuando ya hemos terminado de trabajar, lo que hacemos es indicárselo a git con el comando que aparece en la imagen de a continuación:

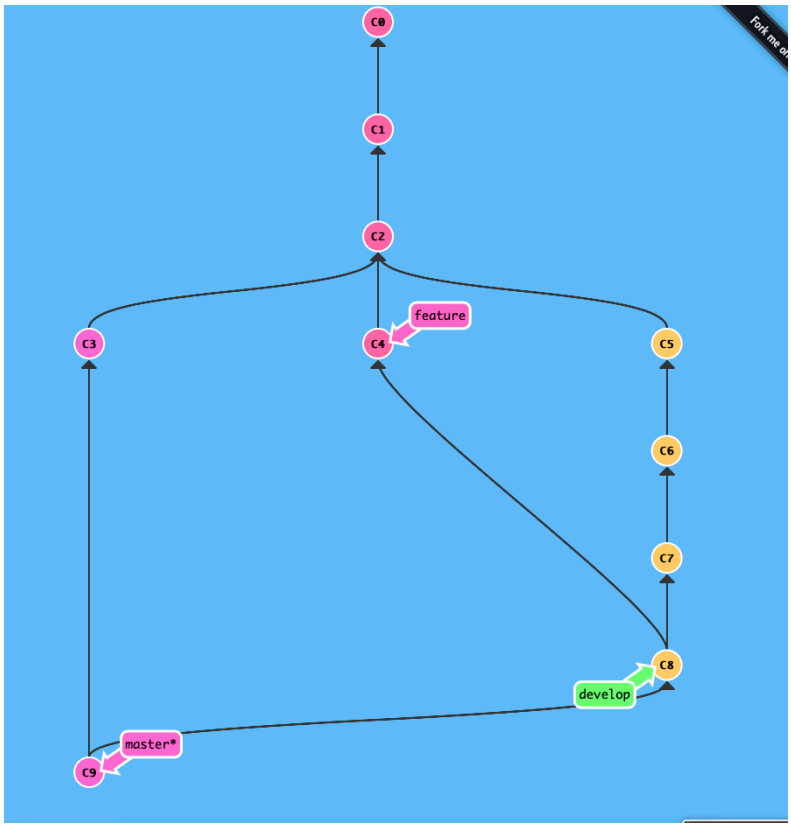
```
Mario@DESKTOP-OIJ9KBH MINGW64 ~/Desktop/universidad/año3/cuatrimestre 2/Desarrol
lo e integracion del software/PR1_DIS (feature/Producto)
$ git flow feature finish Producto
Branches 'develop' and 'origin/develop' have diverged.
And local branch 'develop' is ahead of 'origin/develop'.
Switched to branch 'develop'
Your branch is ahead of 'origin/develop' by 4 commits.
  (use "git push" to publish your local commits)
Already up to date.
Deleted branch feature/Producto (was 6434efa).

Summary of actions:
- The feature branch 'feature/Producto' was merged into 'develop'
- Feature branch 'feature/Producto' has been locally deleted
- You are now on branch 'develop'


Mario@DESKTOP-OIJ9KBH MINGW64 ~/Desktop/universidad/año3/cuatrimestre 2/Desarrol
lo e integracion del software/PR1_DIS (develop)
$
```

Lo que ha sucedido al realizar este comando es que la rama que se había creado en este caso como *feature/Producto* hace *merge* con la rama que están asociadas las *features* de *git-flow* que es la rama de *develop.* Seguidamente se borra automáticamente la rama *feature* cuando esta hace el *merge.*

Aún así, esto solo lo realizamos en contadas ocasiones y era principalmente para explorar las funcionalidades de gitflow. Generalmente, la alternativa era salirnos del sistema de Gitflow y hacer push de la rama directamente para luego realizar la PR como se ha explicado previamente.



Conclusión:

A modo de conclusión hablar de los retos que se nos han presentado como el hecho de tener que trabajar utilizando Github y Git, dos aplicaciones que no estamos muy acostumbrados a manejar, todo esto sumado al hecho de no haber utilizado con anterioridad Java u Eclipse. Pese a todo esto con una buena comunicación y siguiendo una estructura definida desde el principio, hemos sido capaces de entendernos y sobrellevar las situaciones que se nos han ido presentando a lo largo de este proyecto.

Plantear la práctica no supuso mucha complicación ya que algunos miembros del equipo ya habían trabajado con herramientas como Git, por lo cual supimos encaminar el proyecto desde el principio. Algún problema para destacar serían los que mencionábamos al principio de nuestro desarrollo, ya que al comienzo de la práctica no usábamos *git-flow*, pero una vez hablado fue cuestión de familiarizarse con ello y empezar a trabajar con esta metodología, que al fin y al cabo facilitó el desarrollo de este trabajo.

A su vez, cabe destacar la relevancia del uso de herramientas como Git para este tipo de proyectos. La gran mayoría de este trabajo se realizó en un contexto en el cuál todos tenemos que trabajar a distancia y en donde es difícil comunicar qué está haciendo cada uno en cada momento. Por ello, el manejo de las pull request y de las ramas *feature,* fue muy importante para el adecuado desarrollo del proyecto.

Bibliografía:

- *Git-flow cheatsheet.* Recuperado el 3 de marzo de 2020, de https://danielkummer.github.io/git-flow-cheatsheet/

- *Git-git Documentation.* Recuperado el 5 de marzo de 2020*, de* https://git-scm.com/docs/git

- *Tratamientos de fichero XML.* Recuperado el 6 de marzo de 2020, de http://jmoral.es/blog/xml-dom