# Gestión del código fuente

Usaremos un repositorio Git llamado AgoraUS1516 para administrar el código fuente del sistema.

En este repositorio, que se aloja en el servicio de alojamiento de repositorios Git Github, cada subsistema se desarrollará en un repositorio diferente dentro del repositorio global. En el repositorio AgoaUS se recogerá el trabajo realizado y funcional de todos los grupos y su correspondiente integración.

Usar un repositorio global facilita el acceso al código por parte de cada grupo. Podemos obtener el código de otros grupos clonando sus repositorios y haciendo pull, obteniendo así el código más reciente del subsistema en cuestión.

Usar un único repositorio nos facilitaría aun más el acceso al código ya que solo tendríamos que hacer pull de la rama master para obtener el código de los demás subsistemas pero nos encontraríamos con el inconveniente de que cualquier persona podría modificar lo que quisiera, independientemente del grupo en el q se halle.

Este inconveniente no se nos presenta porque sólo tenemos acceso de lectura a los repositorios a los que no pertenecemos.

Tampoco se nos presentará problemas de conflictos con los demás grupos al alojarnos en repositorios diferentes.

Si usásemos un único repositorio tendríamos que crear una carpeta o una rama para cada subsistema con el fin de evitar numerosos conflictos que se presentarían al compartir muchos subsistemas la misma estructura de archivos.

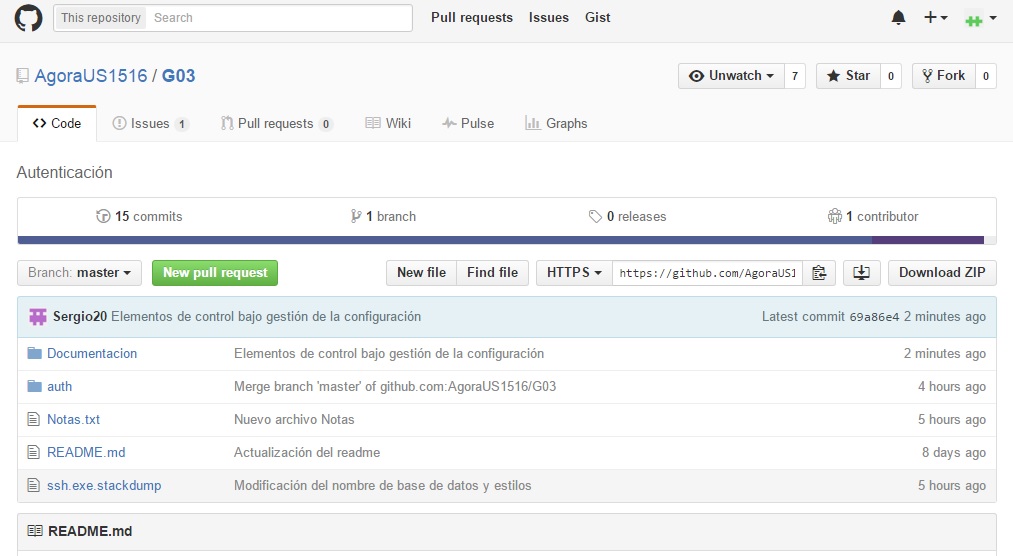


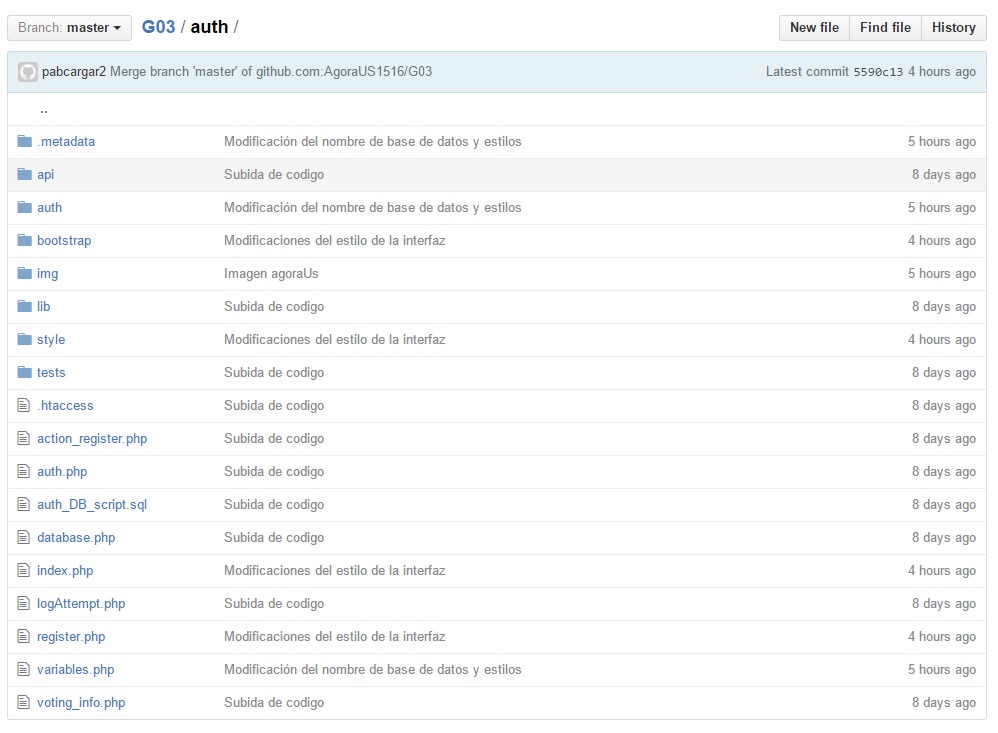
La gestión del código fuente del repositorio de Autenticación se detallará a continuación:

## Arquitectura y creación de documentos

**Abstracción:**

En el directorio principal encontramos la carpeta "auth", que alojará el contenido del subsistema. No se ha escogido una subdivisión en ramas debido al reducido tamaño del subsistema por lo que todas las modificaciones se afectarán en todo el subsistema desde el primer momento. Al realizar modificaciones sobre un proyecto que está en mantenimiento y no en su creación, no necesitamos reutilizar código, lo que también es un motivo para esta decisión.



****

**Implementación:**

Utilizamos la carpeta local que aloja el repositorio en nuestro equipo para la creación y modificación de los ficheros. De este modo para que los cambios sean definitivos se debe realizar commit y luego push. Es responsabilidad individual de cada componente del proyecto asegurarse de que los cambios se hayan realizado correctamente antes de hacer push y de gestionar de forma correcta los conflictos generados.

**Lecciones aprendidas:**

Usar un directorio único para alojar el subsistema nos permitió trabajar de forma más cómoda con la totalidad del proyecto. Si el proyecto estuviese en fase de creación, posiblemente hubiese sido más conveniente utilizar una subdivisión en ramas para poder tener el proyecto actualizado y evitar conflictos pero, debido a que los cambios los realizaremos en partes concretas del proyecto, no lo vemos necesario ya que este hecho minimiza la generación de conflictos y la necesidad de acceder a funcionalidades no realizadas por otros miembros del grupo al encontrarse el proyecto operativo. La desventaja de emplear esta forma de trabajo es que si no hay una comunicación activa y una buena organización puede generar problemas.

**Ejercicio:**

* Enunciado: Una web de venta online se encuentra en un repositorio en Git y se quiere modificar la gestión de las categorías de artículos. Desde que la web se creó no se había realizado ninguna modificación y se van a realizar algunas modificaciones más de la mencionada anteriormente. Además descubrimos que tenemos que modificar un archivo en el que está trabajando un compañero para realizar nuestra modificación. ¿Qué debemos hacer para realizar la modificación correctamente?
* Solución: En primer lugar tendríamos que clonar el repositorio en nuestro equipo. Posteriormente realizamos las modificaciones correspondientes y nos comunicamos con el compañero mencionado en el enunciado para avisar de la modificación que se va a realizar. Una vez todo está en orden hacemos push para que los cambios se apliquen al repositorio global.

## Permisos

**Abstracción:** Todos los miembros del grupo tendrán permisos de lectura y escritura en todo el repositorio.Al ser el equipo y el proyecto reducido vemos innecesario administrar los permisos pues tomaría más tiempo que cualquier problema que pueda ocasionar nuestro método de trabajo.

De esta forma el subsistema también es accesible a todos los miembros del grupo y si hay colaboración, la detección y corrección de errores puede ser más ágil.

**Implementación:** Hemos utilizado GitHub, que nos proporciona repositorios públicos y privados de manera que solo los miembros de un repositorio puedan realizar cambios en este. Pero carece de medidas de seguridad o permisos mas allá de las mencionadas. De este modo hemos utilizado Git de manera habitual intensificando las comunicación entre los miembros del grupo.

**Lecciones aprendidas:** El hecho de no tomar en cuenta los permisos nos ha dado libertad en el acceso al proyecto por lo que no se han dado situaciones de que un miembro del grupo no pueda modificar ciertas partes del proyecto. Esto tiene la desventaja de que cualquiera puede ocasionar un error grave en el proyecto, ya sea de forma maliciosa o no, pero es complicado que suceda esta situación. Al tener sólo los miembros del grupo a nuestro subsistema este riesgo se minimiza notablemente.

**Ejercicio:**

* Enunciado: Un miembro del grupo comprueba que, en un proyecto alojado en un repositorio en Git, falta una modificación menor que debería haber realizado otro miembro del grupo. ¿Cómo debería actuar?
* Solución: Ya que no tenemos restricción de permisos, al ser una modificación menor, si hay compromiso el miembro del grupo debería añadir la modificación que falta.

## Gestión del código en relación al conjunto de subsistemas y nomenclatura

**Abstracción:** En el repositorio común de integración para todos los subsistemas (AgoraUS) se introducirán todos los subsistemas para realizar finalmente su integración. Si los demás grupos necesitan acceso a otros subsistemas, al tener permisos de lectura sobre los repositorios de estos, podrán utilizarlos sin problema. Cuando se implemente la totalidad de la modificación del subsistema se debe hacer push al repositorio AgoraUS. El nombre de cada repositorio será GXX donde XX es el número de grupo. En la descripción de cada repositorio podemos observar el subsistema que aloja.

**Implementación:** Se creó un repositorio GXX por subsistema de manera que todos los grupos tienen permiso de lectura sobre todos ellos pero sólo el grupo encargado de un subsistema tiene permisos de escritura sobre este. También se creó un repositorio de integración común llamado AgoraUS encargado de alojara todos los subsistemas para su integración.

**Lecciones aprendidas:** Debido a esta organización siempre hemos tenido acceso a todo el proyecto y con la seguridad de que sólo el grupo encargado de un subsistema en concreto puede realizar modificaciones sobre este. También la nomenclatura elegida ha hecho más ágil el acceso a las distintas partes del proyecto.

**Ejercicio:**

* Enunciado: Un miembro del grupo es el encargado de revisar el subsistema ya modificado y pasarlo al entorno de integración. Además le falta una última tarea para la que necesita acceder al subsistema del grupo 5. ¿A qué repositorio debe acceder para obtener el subsistema que necesita? Cómo debe actuar para realizar todo esto?
* Solución: En primer lugar tendría que obtener el subsistema que necesita del repositorio G05 para realizar la última tarea y la revisión final del subsistema. Una vez hecho todo esto tendrá que introducir el proyecto en el repositorio de integración AgoraUS.

## Gestión de conflictos

**Abstracción:** En el caso de que se genere un conflicto, será el miembro del grupo que lo divise primero el encargado de resolverlo. Para esto tendrá que ponerse en contacto con los demás miembros o con alguno en concreto si fuere necesario para acordar la versión escogida, etc.

**Implementación:** Utilizamos la consola y el sistema de resolución de conflictos de Git. Este sistema impide un commit o un push en el caso de que haya conflicto además de mostrar un aviso. Para resolver el conflicto, la persona que intentó la operación debe editar manualmente los archivos para quedarnos con el contenido adecuado. Una vez hecho esto, se realiza la operación de nuevo pero sin conflicto.

**Lecciones aprendidas:** La resolución de conflictos ha sido intuitiva y rápida, limitándose a conocimiento del proyecto y comunicación con los demás miembros del grupo.

**Ejercicio:**

* Enunciado: Durante el desarrollo del proyecto, dos componentes del grupo revisan el código buscando errores. Ambos modifican diversos archivos en común de manera que el segundo en subir los cambios provoca un conflicto. ¿Cómo debería actuar dicho miembro?
* Solución: En primer lugar tendría que localizar los conflictos y comprobar si puede resolverlos el sólo en el caso de que sean conflictos menores. Si no fuera esta la situación, tendría que ponerse en contacto con el otro miembro para debatir cual sería la mejor versión final. Una vez hecho esto, el miembro del grupo resuelve el conflicto en función de lo que haya comentado con su compañero y sube los cambios al repositorio.

## Decisión de cambios:

**Abstracción:** Todos los cambios deben ser discutidos y aceptados por los miembros del grupo. Una vez haya quedado clara la forma de proceder, se implementa. La corrección de errores no se comunica a los demás miembros del grupo a no ser que la corrección traiga consecuencias mayores para el proyecto.

**Implementación:** Todos los cambios son discutidos por los miembros del grupo. Casi siempre se llega a un acuerdo rápidamente pero hemos notado que, en ocasiones, tarde o temprano hay que volver a debatir el asunto.

**Lecciones aprendidas:** Esta forma de trabajar nos ha funcionado bien debido a una buena comunicación entre todos los miembros del grupo. No ha supuesto mucho tiempo debatir sobre la mayoría de los cambios exceptuando algunos casos.

**Ejercicio:**

* Enunciado: Un miembro del grupo, revisando el proyecto, se da cuenta de que una funcionalidad del subsistema estaría mejor de otra forma. Además ha encontrado algunos errores en ciertas partes del proyecto. ¿Cómo debería actuar?
* Solución: Debería ponerse en contacto con sus compañeros para debatir el por qué del cambio y la mejor forma de realizarlo. Una vez llegado a un acuerdo, se organiza el cambio para su posterior realización. Con respecto a los errores, en caso de no ser errores de gran envergadura, deberán ser resueltos por el propio miembro del grupo que está revisando. Si se tratase de un error mayor sería conveniente tratar el asunto con los demás miembros.