USANDO GIT

Para aprender a usar GIT se comenzó por realizar el manual http://gitimmersion.com/. y notando que no era suficiente se realizó el curso onlyne [http://keepcoding.es/courses/git-github-sourcetree-agbotraining] y se han hecho multiplex consultas a [http://aprendegit.com/category/general/].

No se ha incluido información sobre la instalación, dado que consiste simplemente en descargarlo desde: https://git-scm.com/downloads, e ir dándole a: siguiente/aceptar.

Introducción

Git es un sistema de control de versiones distribuido, cada uno de los participantes tiene una copia completa del código, los participantes suelen decidir uno de los equipos como repositorio principal y es de este del que se bajan las modificaciones del resto de participantes y suben las suyas. Git genera un árbol de nodos, donde cada nodo es una versión del proyecto que se crea al utilizar el comando commit, no se guardan versiones completas, sino ficheros con instrucciones para convertir una versión del proyecto en otro, reduciéndose enormemente el espacio de disco requerido, además git utiliza enlaces simbólicos para reducir las tareas de creación y eliminación de archivos. Se basa en los comandos de Linux:

- diff archivo1.txt archivo2.txt > diferencias.patch //Generar fichero con las diferencias entre el archivo1 y archivo2, indicando los pasos necesarios a realizar, para que el archivo 1 sea igual al archivo 2.
- patch archivo1.txt < diferencias.patch //Hacer el archivo1 idéntico al archivo 2.

Para trabajar con git es importante tener en cuenta las tres zonas de trabajo:

- Workingcopy: Parte visible del repositorio, que contiene la carpeta de nuestro proyecto.
- Stagin área: Pate solo accesible a través de comandos Git, que contiene los archivos de los cuales se quieren guardar versiones (archivos trackeados).
- Repositorio: Parte solo accesible a través de comandos Git, que almacena la información de los cambios realizados.

Crear/Configurar repositorio

Una vez instalado Git, crear un repositorio es tan sencillo como darle al botón derecho del ratón, dentro de la carpeta en donde están los documentos de los cuales se desea llevar un control histórico de cambios y abrir la consola de Git. Esencialmente para empezar hay que crear el repositorio con el comando: “git init”. Se han añadido también un par de comandos más para configurar el repositorio, como se ve en la siguiente imagen:

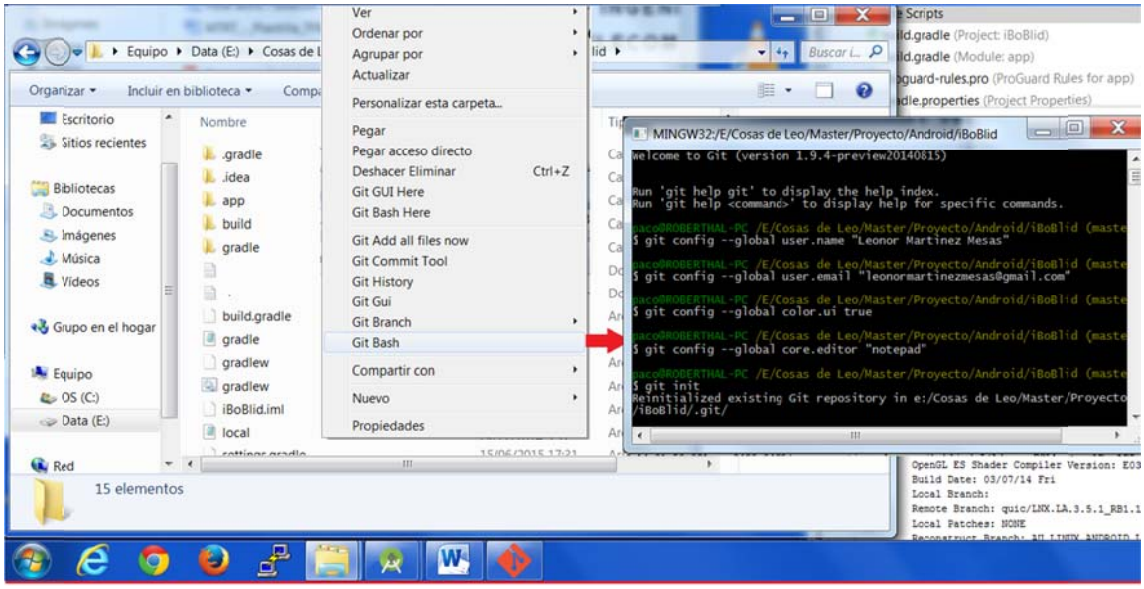

Figura ***¡Error! No hay texto con el estilo especificado en el documento.****.1: Crear un repositorio con Git.*

Git ignore

Lo primero a hacer para cada repositorio, es definir que archivos no se van de trakear, dado que trakear librerías externas o código compilado puede provocar más problemas que beneficios. Pues git no se usa para realizar copias de seguridad del código, sino para tener un control de los cambios que se van realizando y poder deshacer cambios cuando nos damos cuenta, que el código estaba mejor antes. Por lo que solo se ha de trakear el código generado por nosotros. Según la herramienta que estemos utilizando para programar, será necesario indicar uno u otros archivos a ignorar.

Git ignore para Android Studio

Desde Android Studio 1.0, al crear un proyecto se genera automáticamente el fichero Gitignore .

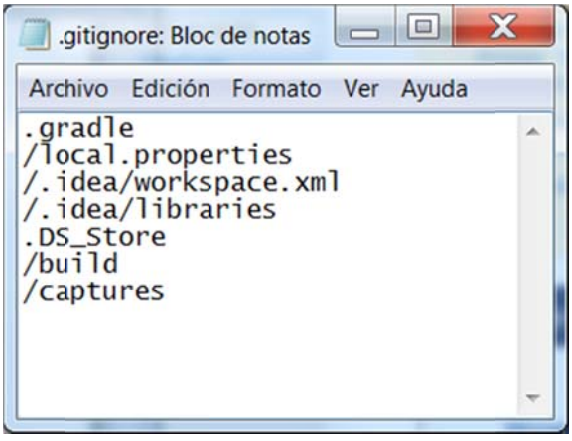

Figura ***¡Error! No hay texto con el estilo especificado en el documento.****.2: Androied Gitignore file.*

Comandos utilizados

Este apartado contiene la chuleta de comandos utilizados en este proyecto para trabajar con GITHUB.

*Tabla **¡Error! No hay texto con el estilo especificado en el documento.**.1: Comandos consola Git.*

Comando	Uso
git config –global core.editor “geditgit”	Configurar el editor de texto
git status	Indica que archivos son trackeados y cuáles no, que archivos están en el staing área y cuáles no, y de los archivos trackeados cuales han sido modificados.
git init	Crear repositorio
git add “archivo”	Pasar un fichero del working copy al Stagin área.
git rm “archivo”	Borra el archivo del “working copy” y añade los cambios al Stagin área.
git commit -m “mensaje”	Generar los ficheros resultantes de hacer diff (hans) entre lo que hay en el repositorio y lo que hay en el Staing area , y almacena en el repo un nuevo nodo compuesto por : un has(identificador unico), los hans y el mensaje. Vacía el Staing área.
git add –A .	“actualizar en el repositorio git el contenido del arrea actual, incluyendo ficheros nuevos o eliminación de ficheros”
git log git log –graph --decorate//ver más claramente las ramas y que commit pertenecen a más de una rama git log –pretty// modo resumen, cada commit en una línea, sin fecha ni autor	Muestra los commit realizados, para cada commit muestra el identificador del commit (has), el autor nombre-email, fecha y el mensaje .
ls -al	Ver todos los archivos con información de permisos y fecha de creación
git diff	Compara contenido del working copy con el staging area
git diff –cached o git diff --stage	Compara el contenido del staging area con el del repo
git diff HEAD	Compara el working copy con el repo
git tag nombreTag	Poner tags, punteros a commits para poder volver a ese punto del proyecto
git tag –d nombreTag	Borrar un tag, un puntero.
git tag	Muestra las tag creadas
git reflog	Nos muestra todos los commit por los que hemos ido pasando, con los identificadores que nos permiten retornar mediante “**git reset –hard identificador**”, a cualquiera de los estados anteriores
git reset HEAD~1	Quitar la acción del ultimo commit, el working copy queda como esta. El staging área queda vació
git reset –hard HEAD~1 //Moverse hacia commit anterires por el log HEAD~x o HEAD^2~x para remas tras un merge ,	Quitar la acción del ultimo commit, modifica el working copy dejándolo tal cual estaba antes del último commit. El

// Moverse hacia commit anterires por el reflog HEAD@[x]	staging área queda vació
git branch	Ver ramas existentes en el repositorio
git branch nombre	Crear una nueva rama
git checkout nombreDeLaRamaOidCommit	Mover el puntero Head, a otrar Rama o a un commit, cambiando el contenido del working copy. Si no movemos a un commit los cambios que ágamos no serán accesibles a no ser que creemos una rama en ese commit y nos movamos a ella, estos dos pasos se pueden hacer en un solo comando:“git checkout –b nombre rama”
git branch –m nuevoNombre	Cambiar nombre de una rama, estando en la rama
git branch –d nombreRama//si estamos en una rama que la alla absorbido git branch –D nombreRama//Borrado de puntero forzoso, pueden quedar commit no apuntados desde ninguna rama	Borrar una rama, hace desaparecer el puntero, pero no se elimina los commit, con esto no cambia ni el working copy ni el staging área, no se puede eliminar la rama en el que uno esta.
git merge nombreRama //Si surgen conflictos, para solucionarlos abrir los archivos en conflicto, observar marcas ==========, parte de arriba, contenido rama actual, parte de abajo contenido rama con la que se hace el merge. Dejar el archivo como se desea y ejecutar “git add archivo”. Una vez modificados todos los archivos en conflicto, realizar un commit para que se ejecute el merge. Si no hay tiempo para esto, abortar mege con “git merge –abort” o echando marcha atras“git reset –hard HEAD~1” o	Merge por defecto con far-forware, La referencia de la rama en la que estoy se mueve a la rama a la que hago el merge. Merge no far-forware, “git merge –no-ff nombreRama”Se crea un nuevo commit con punteros tanto a la rama en la que estoy como a la rema hacia la que ago el merge.
git stash	Guarda en una rama auxiliar el estado del working copy, permitiéndonos cambiar de rama sin necesidad de commitear los cambios realizados en la rama actual. Para recuperar estos cambios hay que ejecutar git stash apply.
git reset HEAD fichero	Quita el fichero del stagin area
git checkout -- fichero	Poner un fichero tal cual estaba en la última versión guardada
git remote	Listar los repositorios remotos, que tenemos asociados al repo en el que nos encontramos.
git remote add nombreRemoteRepo urlRemoteRepo	Añadir un repositorio remoto
git fetch nombreRemoteRepo git	Descargarse un repositorio remoto, crea nuevos punteros a las ramas descargadas para verlos ejecutar “git branch -r”, para hacerlas visibles: 1- ir a la rama con “git checkout repo/rama”, 2-crear rama “git branch rama” o directamente “git checkout rama”.
git branch –u nombreREMOTErepo/ramaRemota ramalocal	Indicarle a git en que rama remota han de subirse ramas locales.

git pull remoteRepo/branch local branch	Este commando es lo mismo que hacer: 1-git fetch remoteRepo/branch 2-git checkout localBranch 3-gir merge remoteRepo/branch
git push nombreRemoteRepo rama git push –f//push forzado, se realiza aunque nuestro repo local, no tenga la última versión del repo remoto	Subir mi repo local al repo remoto
git push nombreRemoteRepo –delete rama	Borrar una rama remota
rm –rf “directorio”	Borrado recursivo forzoso (de todo, incluido ficheros ocultos)
git rebase rama git rebase –onto rama1 rama 2 (coloca los nodos de la rama actual que estan unidos a la rama 2, cortando por la unión sogre la rama 1) git rebase –i (permite cambiar de orden los commit, agruparlos(squash o fixup), editar el texto del commit con reword o incluso colocar el working copy tal cual estaba en un commit concreto, para editarlo de nuevo y guardar dichos cambios(modifivar un commit con edit))	Utilizado para modificar el grafo, deja ocultos los nodos de la rama actual, aciendo una copia de ellos y colocandolos sobre la rama sobre la que se realiza el rebase.
git filter-branch –tree-filter ‘rm –f <filename>’ HEAD	Eliminar un fichero del todo, incluyendo el commit en el que se almaceno, hacer que sea irecuperable. Hay primero que posicionarse en el commit en que se guardó el fichero que se desea borrar con git checkout “id commit”. Aun así sigue siendo accesible usando reflog

REFERENCIAS

[github] https://github.com/LeonorMalaga/i-Boblind.git