**Crear un repositorio Git:**

Dos formas:

1. Dentro de un directorio existente:

git init

1. Clonar un repositorio existente:

git clone [url]. Crea el directorio donde instala el proyecto.

$ git clone git://github.com/schacon/grit.git

Podemos cambiar el nombre del proyecto:

git clone git://github.com/schacon/grid.git ProyectoGrid

**Configuración Git:**

El fichero de configuración global de Git es c:\Usuarios\usuario\.gitconfig

$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

$ git config --global core.editor emacs

$ git config --global merge.tool vimdiff

Para mostrar la configuracion:

git config --list

**Estados de los archivos gestionados con Git:**

- Confirmado (commited):

Los datos están almacenados de manera segura en la DB local.

- Modificado (modified):

El fichero se ha modificado pero no se ha confirmado (commit) a la DB local.

- Preparado (staged):

El fichero modificado ha sido marcado para el siguiente commit.

A un nivel superior existe otra división de los ficheros gestionados por git, tracked o untracked, que quiere decir que son seguidos o no por git.



El directorio de Git es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador.

El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.

El área de preparación es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice, pero se está convirtiendo en estándar el referirse a ella como el área de preparación.

**Comandos git:**

git add <fichero>, dos funciones:

* Incluye el fichero para el seguimiento de git
* Incluye el fichero para el siguiente commit

git branch , lista de branchs disponible en el repositorio local.

git brach –a, lista branchs incluso los ocultos.

git checkout branch1, para cambiar a la rama branch1

git checkout –b branch1, crea la rama branch1 y se coloca en ella.

git commit -m , prepara los ficheros para el push (repositorio remoto y base).

git checkout –- fichero, para recuperar un fichero modificado pero no commiteado, es decir prepardo para el commit.

git commit –a -m , prepara los ficheros para el push aunque no se haya hecho “add“ sobre él.

git config --global –list, configuración global establecida

git config --global --add http.proxy <https://aflucena:PASSWORD@proxy.indra.es:8080>, para definir el proxy en modo global.

git config --global -–unset http.proxy, elemina el poxy global

git config --unset http.proxy, elimina el proxy del repositorio concreto

git config --system --unset credential.helper, para borrar las credenciales (Usuario, password) que almacena window para github.com

git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"

git diff , muestra los cambios de los ficheros no preparados

git diff -- cached , muestra los cambios de los ficheros que están en estado preparado.

git diff –name-status rama1 rama2, muestra diferencias entre dos ramas

git diff rama1:fichero rama2.fichero, muestra las diferencias de un fichero entre dos ramas.

**git fetch branch, crea y carga en rama oculta el branch indicado**

**git help,**  git help <comando>

git log, Muestra las confirmaciones (commits) realizadas, con la opcion –p muestra el detalle.

git merge branch1, hace merge del branch1 sobre la rama activa

git mv <fichero-from> <fichero-to>

raname f1 f2 (Shell dos)

git mv f1 f2, es equivalente a :

git rm f1

git add f2

git pull

git rm <fichero>, para eliminar el fichero y sacarlo del seguimiento git

git rm –f <fichero>, para el eliminar el fichero y sacarlo del seguimiento git cuando tiene cambios.

git rm --cached <fichero>, para sacarlo del seguimiento sin eliminar

git status : muestra los ficheros preparados y no preparados que han cambiado

Existe una aplicación de git para ver visualmente el histórico: gitK

**Carga inicial de un proyecto sobre el repositorio con el branch master en github:**

1. Creamos el repositorio en github:

En el caso de estar el servidor github, podemos seguir las instrucciones y desde la web creamos el repositorio con nombre Proyecto1.

1. En la máquina del desarrollador y en la carpeta donde tenemos el proyecto hacemos :

git init

git add .

git commit -m “primer paso”

git remote add origin <https://github.com/antonio63j/Proyecto1>

antonio63j es el Usuario con la cuenta de github.

git push origin master

**Eliminar un repositorio en github.com:**

https://github.com/*YOUR-USERNAME*/*YOUR-REPOSITORY*/settings

**Colaborar en proyecto github:**

1. fork sobre el repositorio del proyecto donde vamos a colaborar, por ejemplo repositorio antonio63j/SpringRestOneToManyManyToMany. El fork hace ya una copia en nuestro espacio github. Con el usuario github antonio63jun, hacemos el fork.
2. Nos colocamos en nuestro repositorio local, en la carpeta donde queremos colgar el proyecto y clonamos:

git clone <https://github.com/antonio63jun/SpringRestOneToManuManyToMane.git>

1. Si no nos interesan el user y el email globales , los establecemos localmente:

git user antonio63j

git email antonio63j@gmail.com

1. Confirmamos la configuración remota:

$ git remote –v

origin https://github.com/antonio63jun/SpringRestOneToManyManyToMany.git (fetch)

origin https://github.com/antonio63jun/SpringRestOneToManyManyToMany.git (push)

1. Establecemos la ubicación del repositorio original:

$ git remote add upstream https://github.com/antonio63j[/SpringRestOneToManuManyToMany.git](https://github.com/SpringRestOneToManuManyToMany.git)

1. Actualizar la rama master:

si se han hecho cambios nos pedirá hacer commit cuando se intenta el pull

opcion1:

git pull –r upstream master

opcion2:

git fetch upstream

git merge upstream/master

fetch deja los cambios en una rama oculta que modemos ver con “git branch –a”, en este caso sería en upstream/master

o sea que git pull = git fetch + git merge

1. Crear rama desde la rama master para realizar los cambios.

git checkout –b nueva-relacion-entidades

nota: En el repositorio base crear una issue donde se expliquen los cambios que se van a hacer en esta rama.

1. Hacemos commit de los cambios sobre el nuevo branch. Indicamos el repo y el branch con los cambios:

git push origin nueva-relacion-entidades

Se crea el branch nueva-relacion-entidades en github

1. Implementados los cambios, generamos el “ pull request” para que el administrador del repo base lo revise y realice el merge. Desde github, tendremos que pinchar en pull request desde el branch “nueva-relacion-entidades”.

Pruebas realizadas:

Tenemos disponible el repositorio de antonio63j con nombre SpringRestOneToManyManyToMany en el servidor github. Los usuarios de github Antonio63jun y antonio63junio, van a solaparse en las subidas sobre el repositorio master de SpringRestOneToManyManyToMany perteneciente a antonio63j.

En T100, antonio63jun hace fork (click en github.com), crea la Issue3 y clona el proyecto y:

Modifica README y Phone.java. También añade el fichero CambiosIssue3 (sin git add)

$ git config -l

*core.symlinks=false*

*core.autocrlf=true*

*core.fscache=true*

*color.diff=auto*

*color.status=auto*

*color.branch=auto*

*color.interactive=true*

*pack.packsizelimit=2g*

*help.format=html*

*http.sslcainfo=C:/Program Files/Git/mingw32/ssl/certs/ca-bundle.crt*

*diff.astextplain.textconv=astextplain*

*rebase.autosquash=true*

*credential.helper=manager*

*user.name=antonio63j*

*user.email=antonio63j@gmail.com*

*filter.lfs.clean=git-lfs clean -- %f*

*filter.lfs.smudge=git-lfs smudge -- %f*

*filter.lfs.required=true*

*http.proxy=http://aflucena:XXX@proxy.indra.es:8080*

*https.proxy=http://aflucena:XXX@proxy.indra.es:8080*

*core.repositoryformatversion=0*

*core.filemode=false*

*core.bare=false*

*core.logallrefupdates=true*

*core.symlinks=false*

*core.ignorecase=true*

*remote.origin.url=https://github.com/antonio63jun/SpringRestOneToManyManyToMany.git*

*remote.origin.fetch=+refs/heads/\*:refs/remotes/origin/\**

*branch.master.remote=origin*

*branch.master.merge=refs/heads/master*

*user.name=antonio63jun*

*user.email=antonio63jun@gmail.com*

*remote.upstream.url=https://github.com/antonio63j/SpringRestOneToManyManyToMany.git*

*remote.upstream.fetch=+refs/heads/\*:refs/remotes/upstream/\**

En T200, antonio63jun, hace un fetch y un merge :

git fetch upstream

git merge upstream/master

Los resultados son:

No se observan problemas, los cambios en README y Phone.java permanecen.

En T250, hace un pull:

*git pull –r upstream master*

Devuelve este error:

*$ git pull -r upstream master*

*error: cannot pull with rebase: You have unstaged changes.*

*error: please commit or stash them.*

En T251, hacemos un

*git add .*

*git commit -m "Primer commit con cambios en README y Phone.java, ademas se añade CambiosIssue3"*

En T260, hacemos un

git fetch upstream

git merge upstream/master, a este último responde con : Already up-to-date.

A continuación:

*$ git pull -r upstream master*

*From https://github.com/antonio63j/SpringRestOneToManyManyToMany*

*\* branch master -> FETCH\_HEAD*

*Current branch master is up to date.*

En T270 decidimos crear una nueva rama, tal vez hubiese sido lo apropiado justo al principio de clonar el repo.

git checkout –b Issue3

modificamos fichero application-properties

git commit -a –m “para issue3”

En T309, antonio63junio hace fork, crea la issue 4, clona el proyecto y:

334 git add .

335 git config user.name "antonio63junio"

336 git config user.email "antonio63junio@gmail.com"

339 git remote add upstream https://github.com/antonio63j/SpringRestOneToManyManyToMany.git

342 git pull -r upstream master

345 git checkout -b rama-para-issue4

se modifica README

git commit -a -m "modifica AREADME cambiando la fecha de comienzo"

En T400, antonio63jun actualiza el proyecto base:

git push origin rama-issue3

y desde github.com hacemos un pull resquest para que antonio63j, lo incluya en proyecto base.

En T500, antonio63j hace el merge sobre sobre la rama base.

En T00, antonio63junio hace:

git fetch upstream

git merge upstream/master

git config --system --unset credential.helper

git push origin rama-master-issue4

En bithub.com, realizamos el pull request

En definitiva, seguimos los mismos pasos descritos anteriormente (antonio63jun) para actualizar el repositorio base (el de antonio63j en github).

Para actualizar el repositorio base en github.com de antonio63jun, hemos seguidos los pasos descritos en

[http://stackoverflow.com/questions/20984802/how-can-i-keep-my-fork-in-sync-without-adding-a-separate-remote/21131381#21131381](http://stackoverflow.com/questions/20984802/how-can-i-keep-my-fork-in-sync-without-adding-a-separate-remote/21131381%2321131381)., que parece equivalente a eliminar el repositorio y volver a hacer un fork.

Reajuste de los ficheros controlados por git:

1. ajustar fichero .gitignore
2. git rm –-cache –r .

1. git add .
2. ver modificaciones con git status
3. git –m commit