Conceptos básicos

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.

Configuración git

Here is where you will find Git configuration files on Windows:

|  |  |  |
| --- | --- | --- |
| Location of Windows Git Config Files | | |
| Scope | Location and Filename | Filename Only |
| System | <git-install-root>\mingw64\etc\gitconfig | gitconfig |
| Global | C:\Users\username\.gitconfig | .gitconfig |
| Local | <git-repo>\.git\config | config |
| Worktree | <git-repo>\.git\config.worktree | config.worktree |
| Portable | C:\ProgramData\Git\etc\config | config |

As for the location of the Linux Git configuration files, here is where you’ll find them on Ubuntu:

|  |  |  |
| --- | --- | --- |
| Ubuntu Linux Git Config File Locations | | |
| Scope | Location and Filename | Filename Only |
| System | ~etc/gitconfig | gitconfig |
| Global | ~home/<username>/.gitconfig or ~root/.gitconfig | .gitconfig |
| Local | <git-repo>/.git/config | config |
| Worktree | <git-repo>/.git/config.worktree | config.worktree |

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

El fichero de configuración -- system de git es C:\Program Files\Git\mingw32\etc\gitconfig

El fichero local de configuración git está en .git/config dentro de la carpeta del proyecto.

$ 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 configuración:

git config --list o git config --global –list

Para mostrar los ficheros de configuración del proyecto:

git config --list –show-origin

Configuración diff y merge.

Para beyond compare:

git config --global diff.tool bc3

git config --global difftool.bc3.path “C:\App\Beyond Compare/beyond Comapre.exe”

git config --global merge.tool bc3

git config --global mergetool.bc3.path “C:\App\Beyond Compare/beyond Comapre.exe”

git config –global merge.tool bc3

git config –global difftoll.bc3.path “"c:/Program Files (x86)/Beyond Compare 3/bcomp.exe",

git difftool foofile.txt, compara con beyond

Para Meld:

Configuración sobre el fichero de configuración global (c:\Usuarios\usr\.gitconfig

[merge]

tool = meld

[mergetool "meld"]

cmd = \"C:\\Archivos de programa\\Meld\\meld\\meld.exe\" "$LOCAL" "$BASE" "$REMOTE" "--output=$MERGED"

path = C:/Archivos de programa/Meld/meld/meld.exe

trustExitCode = false

keepBackup = false

[diff]

tool = meld

[difftool "meld"]

cmd = \"C:\\Archivos de programa\\Meld\\meld\\meld.exe\" "$LOCAL" "$REMOTE"

path = C:/Archivos de programa/Meld/meld/Meld.exe

[mergetool]

keepBackup = false

To launch a diff using Beyond Compare, use the command "git difftool foofile.txt".

At a Windows command prompt, enter the commands:  
 git config --global merge.tool bc3  
 git config --global mergetool.bc3.path "c:/Program Files (x86)/Beyond Compare 3/bcomp.exe"

To launch a 3-way merge using Beyond Compare, use the command "git mergetool foofile.txt".

Trabajando con repositorios

Crear repositorio en github a partir de proyecto local

* Creamos el repositorio en github, por ejemplo el repositorio con nombre proyecto
* Dentro de la carpeta que incluye el proyecto, podemos hacer:

git init

git init --initial-branch=main (solo a partir de git 2.30, indicamos la rama ‘main’ como default, que también es el repositorio dafault de github.com)

git add .

git commit –m”Primer commit”

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

git push origin master

Nota:

antonio63j es el usuario de la cuenta en github.com

Crear repositorio local desde repositorio base (github.com)

git clone [git@github.com:antonio63j/Proyecto1.git](mailto:git@github.com:antonio63j/Proyecto1.git)

También, git clone <https://github.com/antonio63j/Proyecto1.git>

De esta manera, descargamos el HEAD de la rama master del proyecto.

En caso real, si tenemos en github un repositorio con dos ramas, master y desarrollo, si hacemos el git clone <repositorio>, podemos ver que no tenemos disponible la rama desarrollo:

$ git branch -a

\* master

remotes/origin/HEAD -> origin/master

remotes/origin/desarrollo

remotes/origin/master

Si queremos echar un vistazo a la rama de desarrollo del repositorio base:

git checkout origin/desarrollo.

El indicador de rama en curso de git cliente nos muestra un número:

$ git checkout origin/desarrollo

Note: checking out 'origin/desarrollo'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b <new-branch-name>

HEAD is now at c539c85... se añade la version GitApuntes\_1.0

aflucena@AFLUCENAPW7 MINGW32 /d/temp/pruebasclone/misNotas/software ((c539c85...))

$

Si queremos trabajar en la rama de desarrollo, hacemos

$ git checkout desarrollo

Switched to branch 'desarrollo'

Your branch is up-to-date with 'origin/desarrollo'.

aflucena@AFLUCENAPW7 MINGW32 /d/temp/pruebasclone/misNotas/software (desarrollo)

$ git branch -a

\* desarrollo

master

remotes/origin/HEAD -> origin/master

remotes/origin/desarrollo

remotes/origin/master

aflucena@AFLUCENAPW7 MINGW32 /d/temp/pruebasclone/misNotas/software (desarrollo)

Para añadir una rama nueva a partir de master, por ejemplo la rama arreglo:

git checkout master

git checkout –b arreglo

Para cambiar el nombre del proyecto local desde el comando clone:

git clone <https://github.com/antonio63j/Proyecto1.git> Proyecto

Clone con la opción –b <rama>

Descarga y apunta a la rama indicada.

Tenemos la opción de trabajar con la rama master, para ello “git checkout master”.

¿Qué pasa si hay 2 ramas y además la rama master?

**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 branch –a, lista branchs incluso los ocultos.

git branch –d <local-branch>, elimina la rama local <local-branch>, con –D para forzar el borrado sin chequear el estado del merge.

Para eliminar una rama remota:

git push origin –-delete <remote-branch>,

git checkout branch1, para cambiar a la rama branch1

git checkout –b branch1, crea la rama branch1, copia todos los ficheros de la rama desde donde se lanza este comando y se coloca en ella (en branch1). Equivale a:

git branch branch1

git checkout branch1

git checkout -b desarrollo origin/desarrollo, la idea es crear la rama desarrollo a partir de la rama local oculta origin/desarrollo. Previamente para tener una versión actualizada, tendremos que hacer:

git fetch origin o git fetch origin desarrollo (solo rama desarrollo)

git checkout -b desarrollo origin/desarrollo.

Podremos tener también una rama desarrollo en local desde el repositorio github con:

git clone -b desarrollo <https://github.com/antonio63j/proyecto.git>

CORREGIR, DESHACER

git checkout –- <fichero>, para recuperar los contenidos de un fichero modificado. Si el fichero ya está preparado para commit (hecho add), tendremos que hacer un git reset HEAD <fichero> y luego un checkout -- <fichero>.

CORREGIR, DESHACER

git commit –-amend, sustituye o corrige el último commit. Por ejemplo, si olvidamos incluir en el commit los cambios sobre un fichero, podemos:

git commit –m “commit incompleto”

git add ficheroquefaltaba y que no estaba en estaged (add) cuando se hizo el commit.

git commit --amend

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

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 config –-global push.default, mostrar push.default

git config –-global push.default <tipo para push>, establece el modo en el que se va a elegir la rama para la subida: nothing, current, upstream, matching, simple.

CONSULTA

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

**Para ver las diferencias con respecto al repositorio base, tenemos que hacer un git fetch origin**, que actualizará la rama oculta local con nombre remotes/origin/master.

Ahora sí podremos ver los cambios con git diff master origin/master

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

git diff-tree --no-commit-id --name-only -r <idcommit>, muestra los ficheros confirmados en un commit identificado por <idcommit>.

Nota: para ver los commits realizados: git log [-p]

**git fetch [branch], crea y carga en ramas ocultas las ramas del repositorio, si se indica branch, sólo se carga esa rama (normalmente branch = origin/master)**

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

CONSULTA

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

git log –-branches –-not –-remotes, muestra los commits realizados pendientes de push. A continuación podemos hacer un git show <id-commit> para ver los ficheros implicados en un commit.

git log [–-follow] <fichero>/<path>, muestra los commits realizados sobre ese fichero o path, <fichero> puede ser un nombre de fichero o un template que identifique a un grupo de ficheros. Con –-follow sirve para los renames.

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

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

rename f1 f2 (Shell dos)

git mv f1 f2, es equivalente a :

git rm f1

git add f2

git pull origin master, importar o incorporar los cambios realizados sobre el repositorio remoto. git pull es equivalente a git pull origin master.

git push origin master, subir al repositorio

Importante: push sube ficheros implicados en una confirmación commit, de modo que un fichero que suba al repositorio, podría estar en ese momento en estado modificado, en el caso que se modifique después de un commit. Pero el fichero que se subirá será como estaba en el momento que se commiteo, es decir no se suben los cambios hechos después del commit.

BORRAR, ELIMINAR

git push origin –-delete <remote-branch>, para eliminar la rama <remote-branch> del repositorio origin

git push –-tag origin, sube la etiqueta al repositorio remoto

BORRAR, ELIMINAR

Git remote -v , para ver los repositorios remotos

git remote rm <remote>, para eliminar remote, por ejemplo git remote rm origin

git remote rename <remote-old-name> <remote-new-name>, para cambiar el nombre del remoto

CONSULTA

git remote show [repo-remoto], información sobre el repositorio remoto

CORREGIR, DESHACER

git reset –-hard <hash del commit a recuperar>. Deja el directorio de trabajo local tal y como estaba con el commit indicado. Hay que tener en cuenta que elimina los ficheros creados posteriormente a ese commit indicado, por lo que puede estar bien hacer un push, antes del reset –-hard.

CORREGIR, DESHACER

git reset HEAD <file>, para pasar al estado solo modificado, es decir fuera de los preparados para commit (to unstage). Si lo que queremos es recuperar el contenido de un fichero que hemos modificado tendremos que usar también el comando:

git checkout -- <file>.

BORRAR, ELIMINAR

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, para esto mismo también se puede utilizar git reset HEAD <fichero>

Prueba hecha con rm –-cached <fichero>:

git rm –-cached sobre <fichero> + modificado <fichero> + git push origin master, lo que ha ocurrido es que se ha eliminado <fichero> en el repositorio.

CONSULTA

git show <hash de un commit>:nombre/del/fichero > <fichero destino>, nos muestra el archive en ese momento

git show <id-commit>, información de situación en ese punto.

git show [--stat] --oneline HEAD^^^..HEAD, muestra los cambios en los cuatro últimos commit.

git show --pretty="" --name-only <id-commit>, muestra únicamente los ficheros modificados en el commit.

git diff-tree --no-commit-id --name-only -r <id-commit>, muestra únicamente los ficheros modificados en el commit.

git ls-tree --name-only -r <id-commit>, muestra todos los ficheros implicados en un commit.

también:

git show [--stat] --oneline HEAD

git show [--stat] --oneline <id-commit>

git stash, guarda los cambios (fichero modificados y modificados en estado preparado) en una pila, de forma que puedan ser recuperados. La rama desde donde se hace el stash queda en el estado del último commit. Al intentar incorporar los cambios guardados en la pila podremos (con git stash apply) la necesidad de hacer merge.

git stash list, muestra los stash / pilas stash generadas

git stash apply [stash@{n}] , aplica los cambios guardados, n=0 es última generada

git stash apply –-index, aplica cambios guardados manteniendo el estado de preparado para commit. Si no se indica, todos los ficheros pasan a estado modificado.

git stash branch <nombre-branch>, crea una pila con los cambios, los vuelca en la rama especificada y luego elimina la pila.

git stash drop [stash@{n}], eliminar el stash indicado, por defecto el primero (0).

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

git tag para consultar los tags/etiquetas hechas

git tag –a V0.0.1 –m “message explicativo”, asigna etiqueta sobre el último commit.

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

**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 antonio63jun

git email antonio63jun@gmail.com

1. Consultamos 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

De cualquiera de las dos formas de arriba, se actualize el repositorio local con el repositorio base de github.com.

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

con lo 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:**

**Prueba 1:**

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 fork en github.com), crea la Issue3, 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

**Prueba 2:**

Tenemos el proyecto P en github y hacemos cambios desde dos plataformas de desarrollo pl1 y pl2.

Desde pl2 hacemos cambios en un fichero README, pero no hacemos add ., ni commit

Desde pl1 hacemos cambios en un fichero f1, add ., commint y push al repositorio origin/master de githup.

prueba 2.1:

Si desde pl2 hacemos git diff master origin/master no aparecen diferencias. **Para ver las diferencias tenemos que hacer en pl2 git fetch origin**, que actualizará la rama oculta local con nombre remote/origin/master.

Si ahora hacemos (en pl2) git diff master origin/master, sí aparecen los cambios que realizó pl1 (f1), pero no aparece como diferencia los cambios realizados por pl2 (fichero README), pues no se ha hecho commit.

A continuación se hace merge (en pl2), git merge origin/master, y actualiza f1 en pl2. También vemos que mantiene los cambios realizados en pl2 sobre el fichero README. En este punto git diff origin/master master no presenta diferencias y cuando se hacer el commit en pl2, es cuando si se muestran las diferencias entre las dos ramas, en este caso vemos los cambios del fichero README.

En este punto hacemos git push origin master.

prueba 2.2:

Con el escenario de la prueba2, hacemos cambios en f1 desde pl1, después hacemos git add .; y commit pero al intentar git push origin master, nos da error, advirtiéndonos que debemos integrar los cambios, git nos propone utilizar pull.

A continuación hacemos un git pull origin master, en este proceso se solicita etiquetar un commit para el merge. Finalizado el pull, podemos hacer el git push origin master sin problemas.

prueba 2.3:

Una vez realizados los pasos de la prueba 2, desde pl2 se modifica f1, hacemos git diff master origin/master y no muestra cambios. Hacemos git fetch origin y a continuación vemos que ya aparecen los cambios efectuados en pl1 sobre el fichero f1, pero no el que desde pl2 se ha realizado en f1. Luego hacemos un git push origin y nos devuelve error indicando que debemos integrar los cambios en la plataforma pl2. Hacemos por tanto un git merge origin/master a lo que git responde con una advertencia de que podrían sobre-escribirse los cambios realizados en f1, por lo que ejecutamos git add., git commit –m. Después de esto ya git nos muestra las modificacionesde pl1 sobre f1 y las propias (pl2 sobre f1), también ejecutamos git merge origin/master sin error.

Luego:

git tag -a V0.0.1 -m "primera versión del proyecto"

push --tag origin master

**prueba 3: Escenarios de pruebas push / pull:**

Tres usuarios trabajando sobre el proyecto misNotas en la rama desarrollo y con repositorio en github.com. En estado inicial, los tres tienen el proyecto sincronizado con el repositorio base.

* Antonio63jun:

Hace modificaciones en f1 y f2, después add, commit y push

* Antonio63j:

Modifica f1, add y commit. A continuación hace pull. En el pull falla el automerge, indicando que se arreglen los problemas y se haga commit. El branch de trabajo que indica git es (desarrollo|MERGIN). Antonio63j, hace git mergetool (winmerge), y es configurado por git entrando en esta herramienta sobre el fichero f1. A continuación quitamos las marcas puestas en f1 por git y salimos de la herramienta winmerge.

Se confirma que f2 se ha actualizado según lo previsto. En este momento vemos con git status que tanto f1 como f2 están en estado preparado para commit.

Antonio63j, modifica f2, y al hacer git status vemos:

*On branch desarrollo*

*Your branch and 'origin/desarrollo' have diverged,*

*and have 1 and 2 different commits each, respectively.*

*(use "git pull" to merge the remote branch into yours)*

*All conflicts fixed but you are still merging.*

*(use "git commit" to conclude merge)*

*Changes to be committed:*

*modified: f1.txt*

*modified: f2.txt*

*Changes not staged for commit:*

*(use "git add <file>..." to update what will be committed)*

*(use "git checkout -- <file>..." to discard changes in working directory)*

*modified: f2.txt*

Curiosamente, f2 queda en dos estados, al hacer commit f2 queda en estado modificado. Por tanto se hace git add + commit antes de push.

* Antonio63jun:

Modifica f1 y f2, y en el pull, obtiene error, avisando que f1 y f2 deberían ser sobreescritos en merge.

Git add sobre f1 y en pull da error avisando que f2 debería ser sobreescrito por merge.

Git add sobre f2 y en pull da error avisando que f1 y f2 deberían ser sobreescritos en merge.

Se hace commit y después pull, git avisa que el merge automático sobre f1 y f2 ha fallado y el branch que muestra git es (desarrollo|MERGING). Git sugiere arreglar los ficheros y hacer commit.

Se aplica la herramienta winmerge con git mergetool, que nos va guiando en el merge sobre los dos ficheros. Cuando guardamos f1 y salimos, git arranca winmerge con el f2.

* Antonio63j:

Modifica f1, hace add + commit y con el pull avisa:

*$ git status*

*On branch desarrollo*

*Your branch and 'origin/desarrollo' have diverged,*

*and have 1 and 2 different commits each, respectively.*

*(use "git pull" to merge the remote branch into yours)*

*You have unmerged paths.*

*(fix conflicts and run "git commit")*

*(use "git merge --abort" to abort the merge)*

*Changes to be committed:*

*modified: f2.txt*

*Unmerged paths:*

*(use "git add <file>..." to mark resolution)*

*both modified: f1.txt*

con el pull se han incluido las líneas orientativas en f1.txt y se ha actualizado f2.

Con git mergetool, entra en f1 para eliminar las líneas orientativas del merge.

Se modifica f2, luego add + commit + push

**prueba 4: Escenarios de pruebas push / pull:**

Tres usuarios trabajando sobre el proyecto misNotas en la rama desarrollo y con repositorio en github.com. En estado inicial, los tres tienen el proyecto sincronizado con el repositorio base.

* Antonio63j:

Modifica f1, f2 + add + commit + push

* Antonio63jun:

Modifica f1, y a continuación hace pull mostrando:

*$ git pull*

*remote: Counting objects: 7, done.*

*remote: Compressing objects: 100% (4/4), done.*

*remote: Total 7 (delta 3), reused 7 (delta 3), pack-reused 0*

*Unpacking objects: 100% (7/7), done.*

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

*1bf0449..14c399e desarrollo -> origin/desarrollo*

*Updating 1bf0449..14c399e*

*error: Your local changes to the following files would be overwritten by merge:*

*f1.txt*

*Please commit your changes or stash them before you merge.*

*Aborting*

f2, no se ha actualizado.

Luego add f1, y a continuación hace pull mostrando:

*$ git pull*

*Updating 1bf0449..14c399e*

*error: Your local changes to the following files would be overwritten by merge:*

*f1.txt*

*Please commit your changes or stash them before you merge.*

*Aborting*

f2, no se ha actualizado.

Luego se hace commit y a continuación se hace pull mostrando:

*$ git pull*

*Auto-merging f1.txt*

*CONFLICT (content): Merge conflict in f1.txt*

*Automatic merge failed; fix conflicts and then commit the result.*

f2, se ha actualizado y f1 ha quedado con la marcas de ayuda para el commit.

Luego se hace poolmerge + commit + push

**prueba 5: Comprobar si estamos actualizados con repositorio base:**

Modo 1:

Podemos ver los últimos commit realizados en nuestro repositorio local y comparar con los del repositorio base.

git fetch (para cargar la rama remota)

git log origin/rama -3

git log -3

Podemos comparar las salidas de los dos últimos comando para comparar los check

Modo 2:

git fetch

git diff/difftool branch1 origin/branch1

Con difftool entra en la herramienta correspondiente y muestra uno a uno los ficheros con diferencias. Para ver las diferencias, los ficheros en local tienen que haberse confirmado (implicado en commit).

**prueba 6: pruebas con git mergetool:**

Dos usuarios trabajando sobre el proyecto misNotas en la rama desarrollo y con repositorio en github.com. En estado inicial, los dos tienen el proyecto sincronizado con el repositorio base.

* Antonio63j:

**Modifica f1, f2 + add + commit**

$ **git push**

Nos informa de se han subido 2 objetos.

* Antonio63jun:

**Modifica f1 y f2 + add + commit**

**$ git push**

Username for 'https://github.com': antonio63jun

To https://github.com/antonio63j/misNotas.git

! [rejected] desarrollo -> desarrollo (non-fast-forward)

error: failed to push some refs to 'https://github.com/antonio63j/misNotas.git'

hint: Updates were rejected because the tip of your current branch is behind

hint: its remote counterpart. Integrate the remote changes (e.g.

hint: 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

$ **git pull**

Auto-merging f2.txt

Auto-merging f1.txt

Merge made by the 'recursive' strategy.

f1.txt | 3 +++

f2.txt | 2 ++

2 files changed, 5 insertions(+)

Si hacemos git status, nos muestra que no hay nada que añadir ni que commitear.

**$ git push**, sin problemas en la actualización, se informa de que se han subido dos objetos.

* Antonio63j:

**Modifica f1, f2 y GitApuntes.docx + add . + commit**

**$ git pull**

Auto-merging f2.txt

CONFLICT (content): Merge conflict in f2.txt

Auto-merging f1.txt

CONFLICT (content): Merge conflict in f1.txt

Automatic merge failed; fix conflicts and then commit the result.

quedando en branch (desarrollo|MERGING)

**$git mergetool**

Entramos en la herramienta configurada en git para merge, en este caso meld. Desde esta herramienta, se mergea f1 y de f2. Y al salir:

**$ git status**

On branch desarrollo

Your branch and 'origin/desarrollo' have diverged,

and have 1 and 2 different commits each, respectively.

(use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: f1.txt

modified: f2.txt

se modifica GitApuntes, ahora:

**$ git status**

On branch desarrollo

Your branch and 'origin/desarrollo' have diverged,

and have 1 and 2 different commits each, respectively.

(use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: f1.txt

modified: f2.txt

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: software/GitApuntes.docx

**$ git commit -m"hacemos commit de con dos ficheros confirmados que pendiente de add GitApuntes"**

automáticamente pasa a rama desarrollo

**$ git push**

**$ git status**

Nos muestra que GitApuntes.docx no está en estado staged.

**prueba 7: pruebas con git merge --abort:**

Dos usuarios trabajando sobre el proyecto misNotas en la rama desarrollo y con repositorio en github.com. En estado inicial, los dos tienen el proyecto sincronizado con el repositorio base y en la rama desarrollo.

* Antonio63j:

**Modifica f1, f2 y f3 + add + commit**

**$ git push**

* Antonio63jun:

**Modifica f2 y hacemos pull**

**$ git pull**

Updating d50a77a..5162a59

error: Your local changes to the following files would be overwritten by merge:

f2

Please commit your changes or stash them before you merge.

Aborting

**Confirmamos que no han variado f1, f2 ni f3**

**Se añade f2 al estado de preparado para commmit (git add f2), si volvemos a hacer git pull, el resultado es idéntico al de arriba.**

**Hacemos commit y intentamos un push**

**$ git push**

**Username for 'https://github.com': antonio63jun**

**To https://github.com/antonio63j/misNotas.git**

**! [rejected] desarrollo -> desarrollo (non-fast-forward)**

**error: failed to push some refs to 'https://github.com/antonio63j/misNotas.git'**

**hint: Updates were rejected because the tip of your current branch is behind**

**hint: its remote counterpart. Integrate the remote changes (e.g.**

**hint: 'git pull ...') before pushing again.**

**hint: See the 'Note about fast-forwards' in 'git push --help' for details.**

**La intención es subir los cambios que hemos hecho en f2. Para ello tenemos que hacer pull,tal y como nos dice en el intento de push.**

**$ git pull**

Auto-merging f2

CONFLICT (content): Merge conflict in f2

Automatic merge failed; fix conflicts and then commit the result.

aflucena@AFLUCENAPW7 MINGW32 /d/TestGit/antonio63jun/misNotas (desarrollo|MERGING)

**En esta situación f1 y f3 han sido actualizados y f2 ha quedado con marcas de git:**

<<<<<<< HEAD

antonio63jun, en 2017/10/10 8:16

=======

**con esto de arriba indica que lo que está entre las marcas <<< y === es lo que debe incorporarse debido al propi f2 local y:**

=======

antonio63j, en 2017/10/09 14:00

>>>>>>> 5162a59479019515264f9275d77f7e18314c49a2

Indica lo que entiende que debería incorporar debido al f2 del repositorio.

**$ git status**

On branch desarrollo

Your branch and 'origin/desarrollo' have diverged,

and have 1 and 1 different commits each, respectively.

(use "git pull" to merge the remote branch into yours)

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Changes to be committed:

modified: f1

modified: f3

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: f2

**En esta situación podemos entrar en f2 y asumir las indicaciones de git y después git add f2 o podemos hacer el merge con la herramienta configurada (git mergetool):**

**finalizado el merge, hacemos git status:**

**$ git status**

On branch desarrollo

Your branch and 'origin/desarrollo' have diverged,

and have 1 and 1 different commits each, respectively.

(use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

**modified: f1**

**modified: f2**

**modified: f3**

**Añadimos ahora una modificación en f1**

**$ git status**

On branch desarrollo

Your branch and 'origin/desarrollo' have diverged,

and have 1 and 1 different commits each, respectively.

(use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: f1

modified: f2

modified: f3

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: f1

**Es de esperar que si hacemos commit ahora, no se suba (al hacer push) los últimos cambios hechos en f1.**

**Lo comprobamos, hacemos commit y vemos el nuevo estado:**

**$ git status**

On branch desarrollo

Your branch is ahead of 'origin/desarrollo' by 2 commits.

(use "git push" to publish your local commits)

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: f1

**hacemos push, y confirmamos que los últimos cambios hechos en f1 no han subido.**

**Después del push, f1 sigue en estado modificado, pendiente de incorporar en los preparados para commit.**

**Si nos pasamos al entorno de antonio63j, y hacemos pull vemos:**

**$ git pull**

remote: Counting objects: 6, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 6 (delta 4), reused 6 (delta 4), pack-reused 0

Unpacking objects: 100% (6/6), done.

From https://github.com/antonio63j/misNotas

5162a59..914c2b3 desarrollo -> origin/desarrollo

Updating 5162a59..914c2b3

Fast-forward

f2 | 3 ++-

1 file changed, 2 insertions(+), 1 deletion(-)

**f2 y f3 se incluyeron en el push (de antonio63jun) pero coinciden con los locales de antonio63j, (¿fast-forward indica esto mismo?).**

**antonio63j, modifica f2 y GitApuntes y hace push.**

**Pasamos ahora al entorno de antonio63jun, que hace pull y se queda colgado, no responde, tenemos que hacer un ctr^c para desbloquear, borramos el fichero .git/file-lock y se vuelve a intentar:**

**$ git pull**

remote: Counting objects: 5, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 5 (delta 3), reused 5 (delta 3), pack-reused 0

Unpacking objects: 100% (5/5), done.

From https://github.com/antonio63j/misNotas

9ad0524..5133ee5 desarrollo -> origin/desarrollo

Updating 9ad0524..5133ee5

Fast-forward

f2 | 2 ++

software/GitApuntes.docx | Bin 101513 -> 90556 bytes

2 files changed, 2 insertions(+)

Tras el pull, f1 no se ha modificado y continua es estado pendiente de add.

**antonio63j, modifica f1 y f3 y hace push**

**Pasamos ahora al entorno de antonio63j, que hace pull:**

**$ git pull**

Updating 5133ee5..1047f97

error: Your local changes to the following files would be overwritten by merge:

f1

Please commit your changes or stash them before you merge.

Aborting

**Conclusión, si en el pull exite algún fichero que tenemos modificado en local, no se modifica nada de local.**