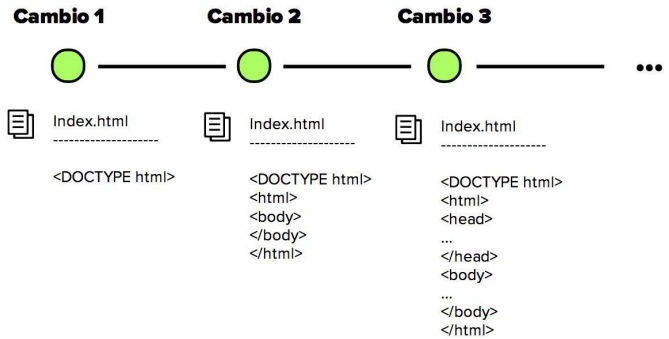
GeeksHubs
academy _

GIT

Píldora Formativa

Control de versiones: GIT

Control de versiones



El control de versiones es un sistema que **registra los cambios realizados** sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que permite recuperar versiones específicas.

Sistema de control de versiones manual

Un **método de control de versiones** usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son avispados).

Éste enfoque es muy común porque es muy simple, pero también tremendamente **propenso a errores**.

Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Historia de Git

Durante la mayor parte del mantenimiento del núcleo de Linux (1991-2002), los cambios en el software se pasaron en forma de parches y archivos.

En 2002, el proyecto del núcleo de Linux empezó a usar un DVCS propietario llamado **BitKeeper**

¿Cuándo surge ?

En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper se vino abajo, y **la herramienta dejó de ser ofrecida gratuitamente**.

Esto impulsó a la comunidad de desarrollo de Linux (y en particular a **Linus Torvalds**, el creador de Linux) a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron durante el uso de BitKeeper.

Objetivos de GIT

- Trabajo en Local
- Agnóstico de la plataforma
- Manejo de ‘deltas’ entre cambios en ficheros
- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Define Flujos de Trabajo en Ramas como metodología (git-flow)
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Cómo funciona

¿Qué es un repositorio Git?

Un repositorio Git es un almacenamiento ‘momentáneo’ de la información del proyecto permitiendo :

- Guardar diferentes versiones del código.
- Acceder a dicha información cuando se necesite.
- Definir ramas de trabajo.
- Borrar información.
- Visualizar gráficas del trabajo
- ...

Estados

Define 3 estados principales en los que se pueden encontrar los archivos:

- Modificado (modified)
- Preparado (staged)
- Confirmado (committed)

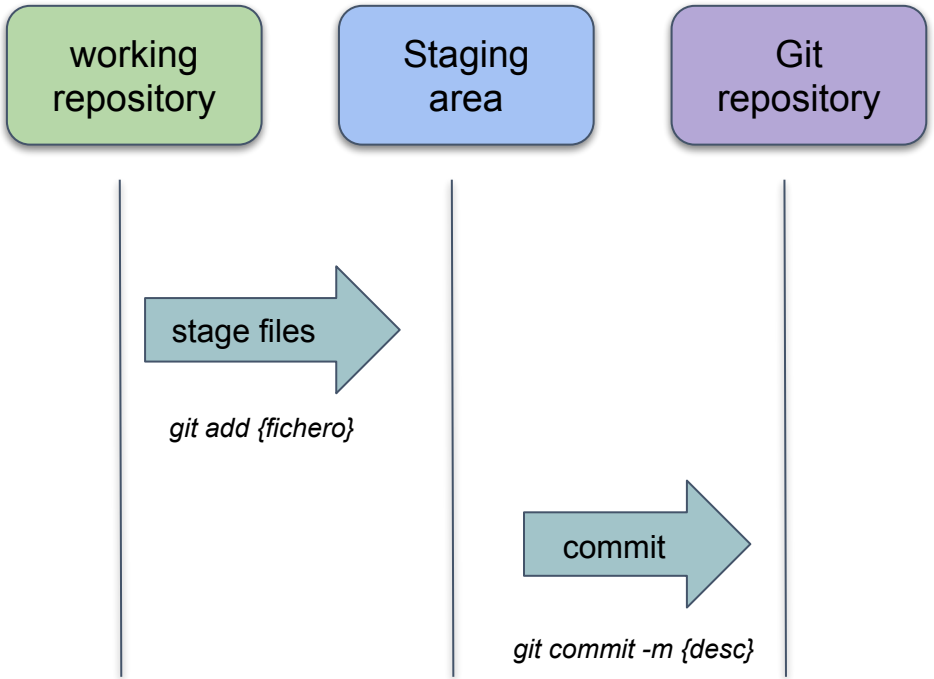
Significado

- **modified** - Estado en el que el/los ficheros han sido **modificado** en el repositorio del proyecto en local (workspace).
- **staged** - Estado donde estos fichero/s modificados se **añaden** a una cola de ficheros ‘pendientes’ de confirmación.
- **committed** - Estado en el que se efectúa el **guardado** de el/los ficheros con una descripción de los cambios en local.

Flujo de trabajo

- Modificar una serie de archivos en el directorio de trabajo (workspace).
- Preparar los archivos a la cola de ficheros pendientes.
  - **git add {fichero}**
- Confirmar los cambios.
  - **git commit -m {descripcion}**

Operaciones locales básicas



Herramienta

¿Qué herramientas utilizamos?

Linux

Windows

Mac

Instalar Herramienta

Linux

Manual -> https://git-scm.com/download/linux

Debian/Ubuntu

apt-get install git

Fedora

yum install git (up to Fedora 21)
dnf install git (Fedora 22 and later)

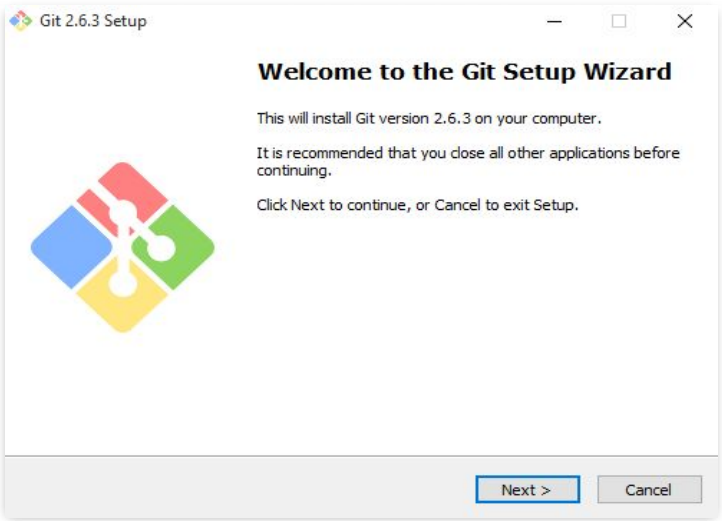
...

Mac

Manual -> https://git-scm.com/download/mac

Windows

Manual -> https://git-scm.com/download/win



Configuración

Primeros pasos

Definir identidad

```
vbolinches@ALF703 MINGW64 ~/Documents/
$ git config --global user.name “vbolinches”

vbolinches@ALF703 MINGW64 ~/Documents/
$ git config --global user.name
vbolinches

vbolinches@ALF703 MINGW64 ~/Documents/
$ git config --global user.email vbolinches@alfatecsistemas.es

vbolinches@ALF703 MINGW64 ~/Documents/
$ git config --global user.email
vbolinches@alfatecsistemas.es
```

Primeros pasos

Editor de texto por defecto

vbolinches@ALF703 MINGW64 ~/Documents/
$ git config --global core.editor code

Primeros pasos

Comprobar configuración

Usar el comando 'git config --list' para listar todas las propiedades que Git define por defecto tras la configuración:

```
vbolinches@ALF703 MINGW64 ~/Documents/
$ git config --list

user.name=Scott Chacon
user.email=schacon@gmail.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

Primeros pasos

Manuales

Si alguna vez se necesita ayuda usando Git, hay tres formas de ver la página del manual (manpage) de cualquier comando:

```
$ git help {comando}
$ git  {comando>} --help
$ man git-  {comando}
```

Uso en local

Inicializando un directorio existente

Para definir un nuevo repositorio Git, se necesita ir al directorio del proyecto y escribir:

```
vbolinches@ALF703 MINGW64 ~/Documents/
$ mkdir gitProject

vbolinches@ALF703 MINGW64 ~/Documents/
$ cd gitProject

vbolinches@ALF703 MINGW64 ~/Documents/gitProject
$ git init
Initialized empty Git repository in C:/Users/vbolinches/Documents/gitProject/.git/

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
```

Inicializando un directorio existente

Este comando define una carpeta oculta llamada ./git

Dentro de ésta, se define la configuración interna del control de versiones que hace referencia a nuestro proyecto.

```
vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ ls -la
total 21
drwxr-xr-x 1 vbolinches 1049089  0 mar.  17 18:29 ./
drwxr-xr-x 1 vbolinches 1049089  0 mar.  17 18:27 ../
drwxr-xr-x 1 vbolinches 1049089  0 mar.  17 18:29 .git/
-rw-r--r--  1 vbolinches  1049089 12 mar. 17 18:29 myfile.txt
```

Añadiendo archivos

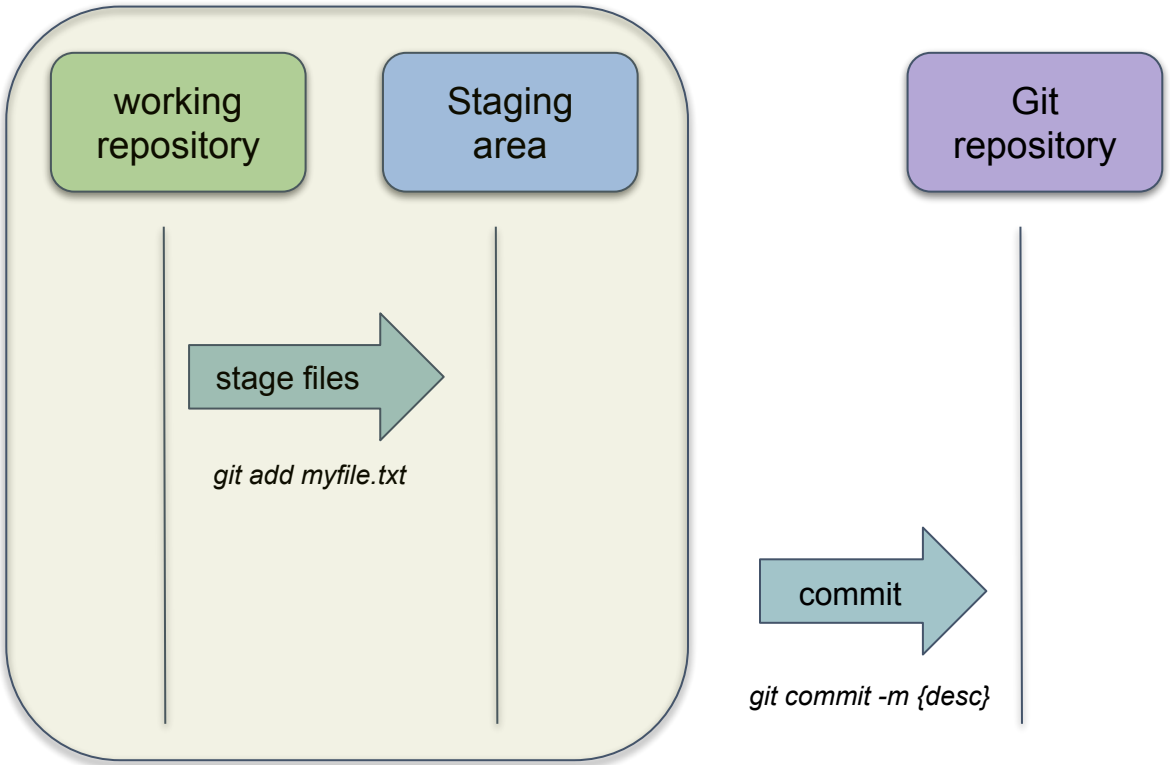
Si se desea empezar a controlar las versiones de los archivos existentes (a diferencia de un directorio vacío), probablemente debas de comenzar a registrar los cambios.

Usa ‘git add {fichero}’ para añadir las modificaciones.

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ echo “hello world” >> myFile.txt

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ git add myfile.txt

Añadiendo archivos



Confirmando cambios

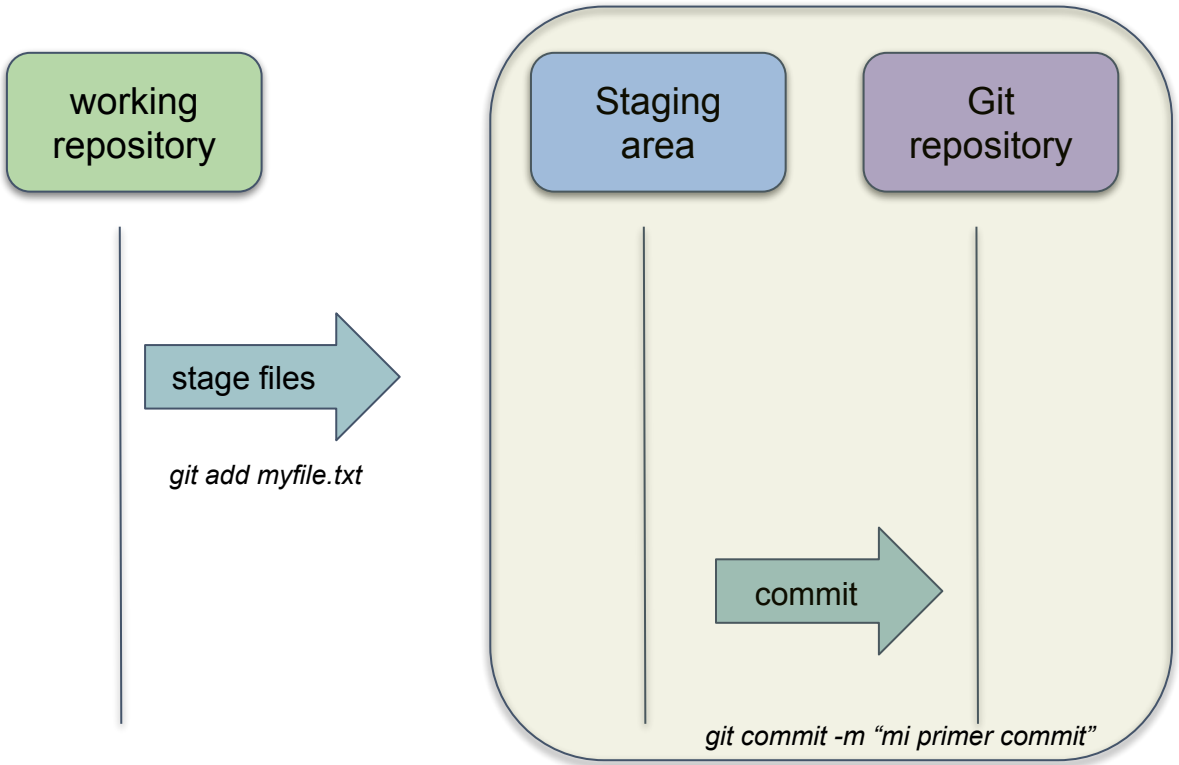
Confirma los cambios con ‘git commit -m {descripción}.

```
vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ echo “hello world” >> myFile.txt

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ git add myfile.txt

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ git commit -m "mi primer commit"
[master (root-commit) 5aeb297] mi primer commit
 1 file changed, 1 insertion(+)
 create mode 100644 myfile.txt
```

Confirmando cambios



Clonando un repositorio existente

Usa ‘git clone {url}’ para realizar una copia exacta del repositorio que necesites.
Por ejemplo, si se quiere clonar la librería Ruby llamada Grit, debes hacer algo así:

```
vbolinches@ALF703 MINGW64 ~/Documents/
$ git clone git://github.com/schacon/grit.git
Cloning into 'grit'...
remote: Enumerating objects: 4051, done.
Receiving obremote: Total 4051 (delta 0), reused 0 (delta 0), pack-reused 4051
Receiving objects: 100% (4051/4051), 2.04 MiB | 7.39 MiB/s, done.
Resolving deltas: 100% (1465/1465), done.
Updating files: 100% (1384/1384), done.

vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
```

Clonando un repositorio existente

Se puede observar que tras clonar el repositorio

- Crea un directorio en local llamado "grit".
- Define una carpeta oculta ‘.git’ dentro de la misma carpeta padre.
- Descarga toda la información de ‘git://github.com/schacon/grit.git’
- Te posiciona en la última versión de la rama Master.
- Si se accede al nuevo directorio ‘grit’, se observan los archivos del proyecto, listos para ser utilizados.

Protocolos de transferencia

Git permite usar distintos protocolos de transferencia.

El ejemplo anterior usa el protocolo git://

También permite usar el protocolo http(s)://

Haciendo uso del usuario y la url podemos usar el protocolo de transferencia SHH de la siguiente manera:

```
$ git usuario@servidor:/ruta.git
```

Comprobando el estado de tus archivos

Para determinar qué archivos están en qué estado se debe usar el comando ‘git status’.

Si se ejecuta este comando justo después de clonar un repositorio, se observa lo siguiente:

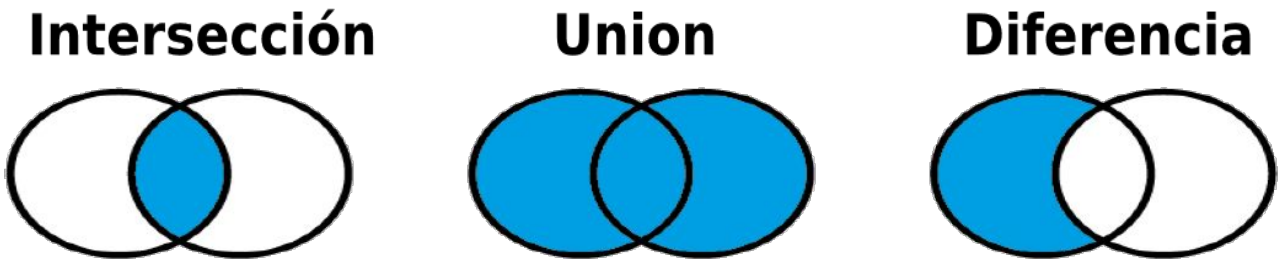
vbolinches@ALF703 MINGW64 ~/Documents
$ cd grit

vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

¿Qué son los deltas?

La eficiencia de calcular solo la diferencia del cambio del fichero respecto al anterior permite a GIT ser tan rápido.



El delta es la intersección de información que solo cambia entre el fichero original y el que lleva los cambios.

Viendo tus cambios

Si se quiere saber exactamente lo que ha cambiado, no sólo qué archivos fueron modificados, puedes usar el comando ‘git diff’.

Dicho comando te muestra exactamente las líneas añadidas y eliminadas.

```
vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ echo "añado al final texto nuevo" >> myfile.txt

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ git diff myfile.txt
The file will have its original line endings in your working directory
diff --git a/myfile.txt b/myfile.txt
index 3b18e51..6118e38 100644
--- a/myfile.txt
+++ b/myfile.txt
@@ -1 +1,2 @@
 hello world
+añado al final texto nuevo
```

Confirmando tus cambios

Confirma los cambios con ‘git commit -m {descripción}.

```
vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ git add myfile.txt
warning: LF will be replaced by CRLF in myfile.txt.
The file will have its original line endings in your working directory

vbolinches@ALF703 MINGW64 ~/Documents/gitProject (master)
$ git commit -m "objetivo cumplido"
[master 0ae1244] objetivo cumplidogit add myfile.txt
 1 file changed, 1 insertion(+)
```

Viendo el histórico de commits

Para comprobar los cambios de nuestro repositorio está el comando '**git log**'

```
vbolinches@ALF703 MINGW64 ~/Documents/test (master)
$ git log
commit 0ae1244a142d4478e70440af0d729c2c1ed93533 (HEAD -> master)
Author: vbolinches <vbolinches@alfatecsistemas.es>
Date:   Tue Mar 17 18:57:16 2020 +0100

    objetivo cumplido

commit 5aeb2974f3daef01daa40798bcfb9129b751738c
Author: vbolinches <vbolinches@alfatecsistemas.es>
Date:   Tue Mar 17 18:39:49 2020 +0100

    mi primer commit
```

Otro comandos de interés

La siguiente lista de opciones interesantes.

- Muestra el mensaje introducido en cada confirmación.
  - -p
- Muestra estadísticas sobre los archivos modificados en cada confirmación.
  - --stat
- Muestra solamente la línea de resumen de la opción.
  - --shortstat
- Muestra la lista de archivos afectados.
  - --name-only
- Muestra la lista de archivos afectados, indicando además si fueron añadidos, modificados o eliminados
  - --name-status
- Muestra una gráfico ASCII con la historia de ramificaciones.
  - --graph

Repositorios remotos

Trabajando con repositorios remotos

Para poder colaborar en cualquier proyecto Git, se necesita saber cómo gestionar el/los repositorios remotos.

Estos son versiones del *workspace* que se encuentran alojados en la nube.

Para realizar cambios en remote se debe usar el comando ‘git push’.

Para recibir las actualizaciones de cambio se debe usar el comando ‘git pull’.

Mostrando tus repositorios remotos

Para ver qué repositorios remotos hay configurados, debes ejecutar el comando `git remote`. Muestra una lista con los nombres de los remotos que hayas especificado.

Añadiendo la opción '-v', muestra la URL asociada a cada repositorio remoto:

```
vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git remote
origin

vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git remote -v
origin  git://github.com/schacon/grit.git (fetch)
origin  git://github.com/schacon/grit.git (push)
```

Añadiendo repositorios remotos

Para asignarle un alias al repositorio remoto de debe usar el comanndo ‘git remote add {nombre} {url}’

```
vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git remote add alias git://github.com/schacon/grit.git

vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git remote
alias
origin

vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git remote -v
alias   git://github.com/schacon/grit.git (push)
origin  git://github.com/schacon/grit.git (push)
```

Recibiendo de tus repositorios remotos

Para recuperar datos de tus repositorios remotos ejecuta:

vbolinches@ALF703 MINGW64 ~/Documents/grit (master) - (fetch + merge)
$ git pull
Already up to date.

vbolinches@ALF703 MINGW64 ~/Documents/grit (master)
$ git fetch

Enviando a tus repositorios remotos

Cuando se han realizado commits en local y se desea sincronizar con el repositorio remoto, se debe de usar el comando ‘git push {nombre-remoto} {nombre-rama}’

vbolinches@ALF703 MINGW64 ~/Documents/Programming-Reactive (develop)
$ git push origin develop:develop
POST git-receive-pack (481 bytes)

remote: Create a pull request for 'develop' on GitHub by visiting:
remote: https://github.com/vicboma1/Reactive-Programming-with-Reactor-Core/pull/new/develop
Branch 'develop' set up to track remote branch 'develop' from 'origin'.
Pushing to https://github.com/vicboma1/Reactive-Programming-with-Reactor-Core.git
To https://github.com/vicboma1/Reactive-Programming-with-Reactor-Core.git
 * [new branch] develop -> develop
updating local tracking ref 'refs/remotes/origin/develop'

Completed successfully.

Enviando forzado a tus repositorios remotos

Puede que a veces la sincronización con el repositorio no sea buena, para hacer un push forzado se debe de usar el comando 'git push --force {nombre-rama:destino-rama}'

vbolinches@ALF703 MINGW64 ~/Documents/Programming-Reactive (develop)
$ git push --force develop:develop
POST git-receive-pack (481 bytes)

remote: Create a pull request for 'develop' on GitHub by visiting:
remote: https://github.com/vicboma1/Reactive-Programming-with-Reactor-Core/pull/new/develop
Branch 'develop' set up to track remote branch 'develop' from 'origin'.
Pushing forced to https://github.com/vicboma1/Reactive-Programming-with-Reactor-Core.git
To https://github.com/vicboma1/Reactive-Programming-with-Reactor-Core.git
 * [new branch] develop -> develop
updating local tracking ref 'refs/remotes/origin/develop'

Completed successfully.

Eliminando y renombrando repositorios remotos

Para renombrar una referencia a un repositorio remoto, en versiones recientes de Git puedes ejecutar ‘git remote rename {descripcion}’.

vbolinches@ALF703 MINGW64 ~/Documents/grit (develop)
$ git remote
alias
origin

vbolinches@ALF703 MINGW64 ~/Documents/grit (develop)
$ git remote rename alias develop

vbolinches@ALF703 MINGW64 ~/Documents/grit (develop)
$ git remote
develop
origin

Trabajando en ramas locales

Como crear y trabajar con ramas

Para definir un nuevo flujo de trabajo fuera de la rama ‘master’ se debe hacer un copia de la misma.

Es necesario definir el nombre de la nueva rama con la nueva funcionalidad.
En ésta aplicaremos los nuevos cambios.

A esta acción se le atribuye el nombre de ‘git branch switching’.

Identificando la rama actual

Para identificar la rama actual en la que se reside se debe de utilizar el comando '**git branch**'

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git branch
* master

Cómo generar una nueva rama

Para identificar la rama actual en la que se reside se debe de utilizar el comando '**git branch {desciption}**'

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git branch develop-nuevaFuncionalidad

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git branch
  develop-nuevaFuncionalidad
* master
```

Cambiando a la nueva rama

Para proceder a un cambio de rama se debe de usar el comando la rama actual se debe de utilizar el comando '**git checkout {rama}**'

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git checkout develop-nuevaFuncionalidad
Switched to branch 'develop-nuevaFuncionalidad'
M       .idea/compiler.xml
M       .idea/encodings.xml
M       .idea/misc.xml

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop-nuevaFuncionalidad)
```

Cambiando a la nueva rama rápidamente

Para proceder a un cambio de rama en un solo comando se debe de utilizar '**git checkout -b {rama}**'

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop-nuevaFuncionalidad)
$ git checkout -b develop-ultima
Switched to a new branch 'develop-ultima'

vbolinches@ALF703 MINGW64 ~/Documents/Reactive (develop-ultima)
```

Mezclando ramas con git merge

Para llevar los cambios de la rama actual a la rama master, primero se debe cambiar a la rama ‘master’.

Seguidamente, usar el comando ‘git merge {rama-con-cambios}

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop-nuevaFuncionalidad)
$ vi pom.xml

** Quitamos 3 líneas con espacios.

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop-nuevaFuncionalidad)
$ git add pom.xml

Mezclando ramas con git merge

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop-nuevaFuncionalidad)
$ git commit -m "quitando espacios en blanco"
[develop-ultima 52557f0] quiando espacios en blanco
 1 file changed, 1 insertion(+), 3 deletions(-)

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop-nuevaFuncionalidad)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git merge develop-nuevaFuncionalidad
Updating f774111..52557f0
Fast-forward
 pom.xml | 4 +---
 1 file changed, 1 insertion(+), 3 deletions(-)
```

Comprobando conflictos después de un merge

¿Qué pasa si se edita el mismo archivo en ambas ramas y sobre las mismas líneas?.

Se generará un conflicto que se debe de resolver a mano.
Git aunque es útil y potente, no tiene la capacidad de decidir qué versión del código en conflicto es la correcta.

Revertir un commit antiguo

Si se desea borrar el último commit en remote se debe de usar el comando ‘git revert {hash}’.

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop)
$ git log
commit 9a6d291614effba613328045f33551db738dd388 (HEAD -> develop)
Author: vbolinches <vbolinches@alfatecsistemas.es>
Date:   Tue Mar 17 20:45:20 2020 +0100 - guau-guau

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (develop)
$ git revert f964a2d9d702261a249032ada89705d1783e68bd
develop  Revert "guau-guau"
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Deshaciendo cambios

Modificando tu último commit

Para realizar una modificación al tu último mensaje de tu commit se debe de utilizar el comando 'git commit --amend'. Éste permite sobreescribir el mensaje del último commit.

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ echo "Guau-guau" > nuevoFichero.txt

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git add nuevoFichero.txt

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git commit -m "Mensaje miau-miau"
[develop 227e27d] subiendo miau-miau
 1 file changed, 1 insertion(+)
 create mode 100644 nuevoFichero.txt
```

Modificando tu último commit

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git commit -amend
error: did you mean `--amend` (with two dashes)?

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git commit --amend -m "Mensaje guau-guau"
[develop 9a6d291] guau-guau
 Date: Tue Mar 17 20:45:20 2020 +0100
 1 file changed, 1 insertion(+)
 create mode 100644 nuevoFichero.txt
```

Cómo podemos deshacer un Merge

En caso de que se haya realizado un merge por error, se puede deshacer el mismo mediante el comando **'git reset'**

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git reset
Unstaged changes after reset:
No files

** En este caso, el control de versiones no ha dejado deshacer el merge...

Forzando el deshacer de un Merge

En casos excepcionales, sobre todo trabajando con la rama master, es necesario aplicar un parámetro para poder forzar el borrado. Se debe usar el comando 'git reset --hard'.

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git reset --hard
HEAD is now at 52557f0 quiando espacios en blanco

Cómo eliminar ramas

Para eliminar las ramas existentes, debemos salir de la misma.
Luego se aplica el comando 'git branch -d {rama}'.

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git branch
  develop
  develop-nuevaFuncionalidad
  develop-ultima
* master

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git branch -d develop-ultima
Deleted branch develop-ultima (was 52557f0).
```

Quitar archivos del stage

Para sacar del stage los archivos añadidos, se debe de usar el comando ‘git reset {fichero}’.
El estado se estos debe de pasar a ‘modified’.

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ echo "nuevo fichero!" > nuevoFichero.txt

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git add nuevoFichero.txt

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git status
On branch develop
Your branch is up to date with 'origin/develop'.
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   nuevoFichero.txt
```

Quitar archivos del stage

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git reset nuevoFichero.txt

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git status
On branch develop
Your branch is up to date with 'origin/develop'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        nuevoFichero.txt
        target/

nothing added to commit but untracked files present (use "git add" to track)
```

Vaciando el stage

Para vaciar el stage, se debe de usar el comando ‘git reset’ en la raíz del proyecto.

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git reset

Almacenando cambios

Guardar cambios

Una vez realizado el stage de los ficheros, se puede almacenar dicha información para realizar un cambio de rama. Este almacenamiento se debe hacer con el comando 'git stash'.

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git add nuevoFichero.txt

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash
stash@{0}: WIP on develop: 9a6d291 guau-guau
```

Listar cambios

Para visualizar los stashes almacenador en la rama, se debe de utilizar el comando ‘git stash --list’.

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash list
stash@{0}: WIP on develop: 9a6d291 guau-guau
stash@{1}: WIP on develop: 987u765 miau-miau

Aplicando cambios conservando el stash

Para aplicar los stashes, conservando el mismo, se debe de usar el comando 'git stash apply --index {id}'. Por defecto, id = 0

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash apply --index 1
<stdin>:8: new blank line at EOF.
+warning: 1 line adds whitespace errors.
On branch develop
Your branch is ahead of 'origin/develop' by 1 commit.
Changes to be committed:
        modified:   nuevoFichero.txt
Untracked files:
        target/
```

Aplicando cambios y eliminando el stash

Para aplicar los stashes y borrando el mismo, se debe de usar el comando ‘git stash pop --index {id}’. Por defecto, id = 0

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash pop --index 1
<stdin>:8: new blank line at EOF.
+warning: 1 line adds whitespace errors.
On branch develop
Your branch is ahead of 'origin/develop' by 1 commit.
Changes to be committed:
        modified:   nuevoFichero.txt
Untracked files:
        target/
```

Eliminando el stash

Para eliminar los stashes y sin tener que aplicarlo se debe de usar el comando ‘git stash drop --index {id}’. Por defecto, id = 0

```
vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash list
stash@{0}: WIP on develop: 9a6d291 guau-guau

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash drop
Dropped refs/stash@{0} (508864ce297e520a2c77f61902e51aa4c34b5def)

vbolinches@ALF703 MINGW64 ~/Documents/Reactive-Programming (master)
$ git stash list
```

RETO con Git
Level 1

Paso 1 - Crear un repositorio git en local

Para definir un nuevo repositorio Git, se necesita...

- Crear una carpeta con el nombre del proyecto
- Acceder a la carpeta
- Inicializar git con el comando: git init
- Validar que se ha creado la carpeta oculta '.git'
- Validar que se reside en la rama 'Master'

Paso 2 - Comandos básicos

Se necesita crear un fichero y manipularlo, para ello se debe:

- Crear un fichero
- Añadir contenido
- Añadir el fichero al estado de ‘Stage’
- Ver el estatus del repositorio
- Validar los cambios con un commit en la rama **develop**

Paso 3 - Mergea cambios a Master

Es preciso subir los cambios a la rama Master, pues debemos trabajar siempre en ramas anexas a ésta. Para ello es necesario mergear Develop en Master.

- Valida que la rama actual es ‘Develop’
- Lista las ramas y cambia a ‘Master’
- Validar que la rama actual es ‘Master’
- Mergea la rama ‘Develop’ a Master
- Valida que los cambios de la rama ‘Develop’ están en ‘Master’

Investigación Tecnológica

Búsqueda de Cultura

Recuerdas el nombre del desarrollador que creó GIT ?

Sabías que es la misma persona que desarrolló el sistema operativo Linux ?

Investiga dónde estudió, con quién trabajó...

Qué otros proyectos de renombre tiene ?

Le ponemos cara ?

Linus Benedict Torvalds

Búsqueda de nuevas Herramientas

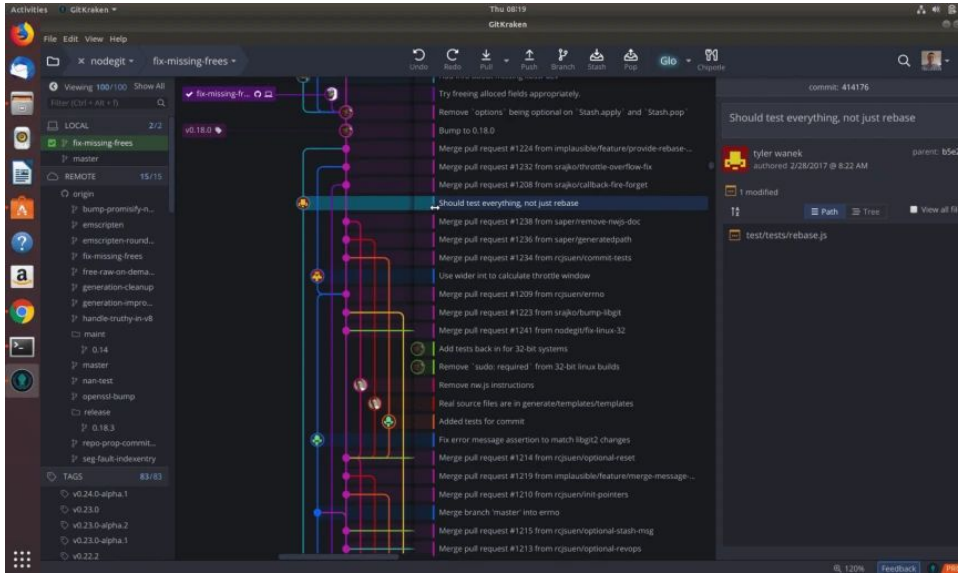
Git al ser agnóstico de la plataforma, puede ser usado en todos los sistemas operativos.

Éste define un manejo más fácil con el uso de una interfaz de usuario.

A continuación, se detallan alguna aplicaciones que debes investigar....

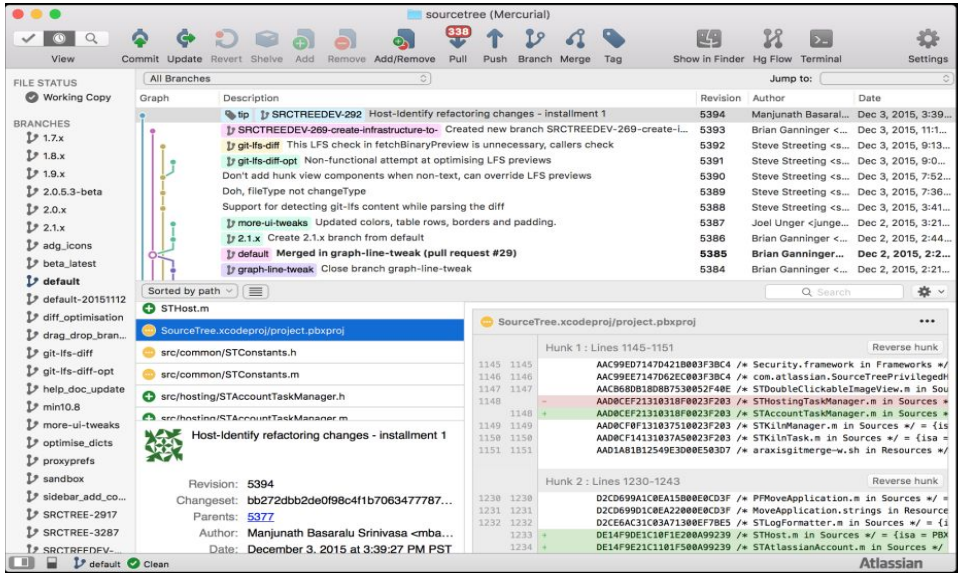
Linux

GitKraken



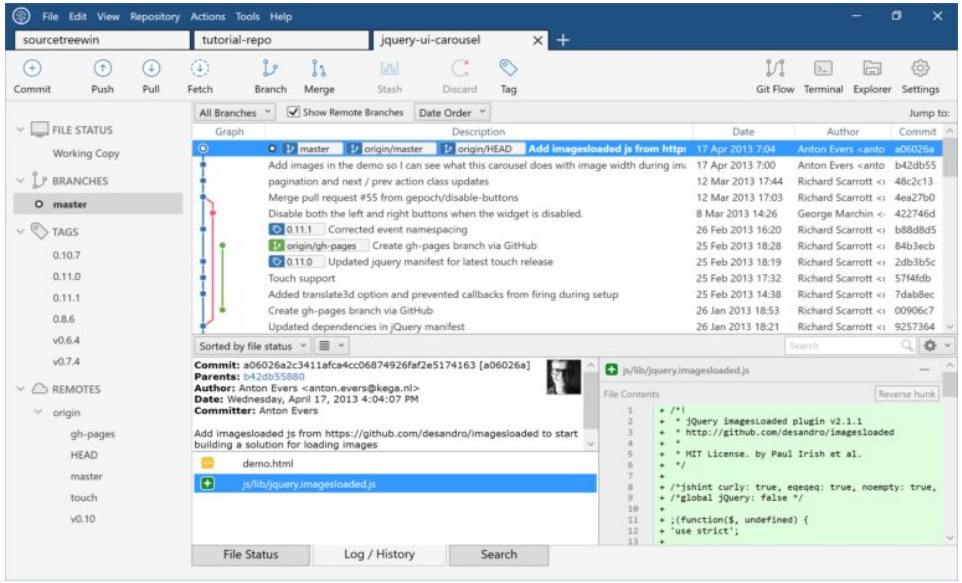
Mac

SourceTree



Windows

SourceTree



Tú decides!