Manual de Git

Primera configuración de Git, primeros comandos que debes lanza

Antes que nada, inmediatamente después de instalar Git, lo primero que deberías hacer es lanzar un par de comandos de configuración.

git config –global user.name "Tu nombre aquí"
git config –global user.email "tu_email_aquí@example.com"

Con estos comandos indicas tu nombre de usuario (usas tu nombre y apellidos generalmente) y el email. Esta configuración sirve para que cuando hagas commits en el repositorio local, éstos se almacenen con la referencia a ti mismo, meramente informativa. Gracias a ello, más adelante cuando obtengas información de los cambios realizados en el los archivos del "repo" local, te va a aparecer como responsable de esos cambios a este usuario y correo que has indicado.

Operativa básica con Git

Ahora vamos a explicar los procedimientos básicos en la operativa del día a día con Git. Acciones como añadir código al repositorio, compartir el código a través de servidores remotos donde nos permitan sincronizar el proyecto entre los integrantes de un equipo y evitar que Git tenga en cuenta archivos que no deberían estar en el proyecto.

Crear una carpeta para tu proyecto y colocar archivos

Crea una carpeta en tu ordenador donde vamos a crear todos nuestros repositorios, puedes crear una carpeta específica para tu primer proyecto con Git. La carpeta de tu proyecto la puedes crear con el explorador de archivos o si quieres por línea de comandos, es indiferente.

A partir de ese momento tienes todo un repositorio de software completo en tu máquina del proyecto que tenías en ese directorio. Puedes crear los archivos que quieras en el directorio (tu proyecto) y luego puedes enviarlos al repositorio haciendo un "commit".

Una vez creada tu carpeta puedes crear archivos dentro. Como estamos hablando de desarrollo de software, los archivos que meterás dentro de tu carpeta contendrán el código de tu aplicación, página web, etc

Inicializar el repositorio Git

Para crear tu repositorio Git, donde monitorizamos los archivos de un proyecto, tienes que realizar una operación previa. Es la inicialización del repositorio Git que queremos crear en la carpeta de tu proyecto y es una operación que deberás realizar una única vez para este proyecto.

Así pues, antes que nada (antes de enviar cualquier archivo al repositorio), creo el repositorio Git con el comando "git init".

git init

Una vez has inicializado el repositorio, podrás observar que se ha creado una carpeta en tu proyecto llamada ".git" . Si ves esa carpeta en tu proyecto es que el repositorio se ha inicializado correctamente y que ya tienes convertida esa carpeta en un repositorio de software Git.

En la carpeta .git es donde se va a guardar toda la información relativa al repositorio. Resulta obvio, pero conviene decir que nunca se debe borrar esa carpeta y tampoco deberíamos acceder de forma directa a sus contenidos, ya que nos vamos a comunicar siempre con el repositorio por medio de comandos.

Guardar los archivos en el repositorio (commit)

Una vez que crees tus primeros archivos, puedes comenzar a trabajar con Git, enviando esos ficheros al repositorio. Aunque los archivos estén en la carpeta de tu proyecto y hayas iniciado el repositorio Git previamente, el sistema de control de versiones no los está monitorizando, por lo que a nivel de tu repositorio Git todavía es como si no estuvieran.

A esa acción de guardar los archivos en el repositorio se llama, en la jerga de Git, hacer un "commit". En este caso el commit lo estás haciendo en local, porque los archivos los estás enviando a tu repositorio local que tienes en tu máquina.

Un commit en Git se hace mediante dos pasos.

1) Tengo que añadir el fichero a una zona intermedia temporal que se llama "Zona de Index" o "Staging Area" que es una zona que utiliza Git donde se van guardando los ficheros que posteriormente luego se van a hacer un commit.

Cualquier archivo que quieras mandar a la zona de index lo haces con el comando "add".

git add nombre-del-fichero
git add .

Una vez añadido podrías ver que realmente tienes ese fichero en el "staging area", mediante el comando "status".

git status

Verás que, entre las cosas que te aparecen como salida de ese comando, te dice que tienes tu fichero añadido como "new file". Además te dice que estos cambios están preparados para hacer commit.

2) Tengo que enviar los archivos de la zona de Index al repositorio, lo que se denomina el commit propiamente dicho. Lo haces con el siguiente comando:

git commit -m "mensaje para el commit"

Con esto ya tenemos nuestro primer archivo dentro del repositorio. A partir de aquí podremos mantener y controlar los cambios que se hagan sobre este archivo (u otros que hayamos incluido por medio del commit).Si haces de nuevo un comando "**git status**" podrás comprobar que no hay nada para enviar al repositorio.

Comando git log

El comando que podemos usar para ver el histórico de commits, estando situados en la carpeta de nuestro proyecto, es:

git log

Como puedes observar, el listado de commits está invertido, es decir, los últimos realizados aparecen los primeros. De un commit podemos ver diversas

informaciones básicas como: Identificador del commit Autor Fecha de realización Mensaje enviado Podría ser algo como esto:

commit cd4bcc8bad230f895fcadd5baf258715466a8eaf
Author: Miguel Angel Alvarez
Date: Fri Feb 10 18:38:41 2017 -0200
ejemplos clase 1 polymerfire

Log en una línea por commit

Es muy útil lanzar el log en una sola línea, lo que permite que veamos más cantidad de commits en la pantalla y facilita mucho seguir la secuencia, en vez de tener que ver un montón de páginas de commits

Para ello usamos el mismo comando, pero con la opción "--oneline":

git log --oneline

Y con la opción –graph lo vemos de forma grafica

git log --graph --oneline

Ver un número limitado de commits

Si tu proyecto ya tiene muchos commits, decenas o cientos de ellos, quizás no quieras mostrarlos todos, ya que generalmente no querrás ver cosas tan antiguas como el origen del repositorio. Para ver un número de logs determinado introducimos ese número como opción, con el signo "-" delante (-1, -8, -12...).

Por ejemplo, esto muestra los últimos tres commits:

git log -3

Ver información extendida del commit

Si queremos que el log también nos muestre los cambios en el código de cada commit podemos usar la opción -p. Esta opción genera una salida mucho más larga, por lo que seguramente nos tocará movernos en la salida con los cursores y usaremos CTRL + Z para salir.

git log -2 -p

Ver información extendida del commit

Si queremos que el log también nos muestre los cambios en el código de cada commit podemos usar la opción -p. Esta opción genera una salida mucho más larga, por lo que seguramente nos tocará movernos en la salida con los cursores y usaremos CTRL + Z para salir.

git log -2 -p

Eso nos mostrará los cambios en el código de los últimos dos commits. Sin embargo, si quieres ver los cambios de cualquier otro commit que no sea tan reciente es mucho más cómodo usar otro comando de git que te explicamos a continuación "git show".

Comando "show": obtener mayor información de un commit

Podemos ver qué cambios en el código se produjeron mediante un commit en concreto con el comando "show". No es algo realmente relativo al propio log del commit, pero sí que necesitamos hacer log primero para ver el identificador del commit que queremos examinar.

Realmente nos vale con indicarle el resumen de identificador del commit, que vimos con el modificador --oneline.

git show b2c07b2

Podrás observar que al mostrar la información del commit te indica todas las líneas agregadas, en verde y con el signo "+" al principio , y las líneas quitadas en rojo y con el signo "-" al principio.

Modificar el último commit, con Git

Una operativa corriente del mantenimiento de un repositorio Git consiste en modificar el commit realizado por último con la opción --amend.

En el día a día de Git hay una situación que se produce bastante a menudo y que podemos solucionar de una manera elegante con una opción determinada de la

acción de commit. Básicamente se trata de modificar el último commit, en lugar de crear uno nuevo.

Imagina que estás editando tu proyecto, realizas una serie de cambios y luego haces el correspondiente commit para confirmarlos en el repositorio. Casi enseguida te das cuenta que cometiste un error. Entonces puedes arreglar tu código para solucionarlo y lanzar otro commit para confirmarlo. Está todo bien, lo que pasa es que la funcionalidad que has agregado es solo una, lo que pasa es que se generaron dos commits cuando lo más elegante es que hubiéramos creado solo uno, con la funcionalidad bien realizada.

La manera de solucionar esta situación es muy sencilla y merece la pena que la veamos, porque no supone ningún esfuerzo y nos permite gestionar nuestro repositorio un poco mejor.

Modificar el commit con --amend

Entonces, vamos a alterar el commit anterior, ya que es lo lógico, en vez de producir un commit nuevo. Simplemente indicamos la opción --amend.

git commit --amend

Recomendado

git commit --amend -m 'Creado el readme editado el commit primero

No te olvides de hacer el "git add .", porque si no en el commit no habrá nada que enviar. En este caso no ponemos mensaje, ya que el mensaje ya lo habíamos indicado en el anterior commit. No obstante, Git nos abrirá un editor para que nosotros podamos editar el mensaje y cambiarlo por el que nos guste más. Puedes ahorrarte el (para algunos fastidioso) editor de línea de comandos indicando el mensaje en el commit con "-m". Aunque estés haciendo un "amend" y ya tengas mensaje lo puedes sobreescribirlo directamente sin tener en cuenta lo indicado anteriormente: git commit --amend -m 'Creado el readme editado el commit primero.

Archivo .gitignore

Qué es el archivo gitignore, para qué sirve, cómo implementar el gitignore en un repositorio Git.

Git tiene una herramienta imprescindible casi en cualquier proyecto, el archivo "gitignore", que sirve para decirle a Git qué archivos o directorios completos debe ignorar y no subir al repositorio de código.

Su implementación es muy sencilla, por lo que no hay motivo para no usarlo en cualquier proyecto y para cualquier nivel de conocimientos de Git que tenga el desarrollador. Únicamente se necesita crear un archivo especificando qué elementos se deben ignorar y, a partir de entonces, realizar el resto del proceso para trabajo con Git de manera habitual.

Implementar el gitignore

Como hemos dicho, si algo caracteriza a gitignore es que es muy fácil de usar. Simplemente tienes que crear un archivo que se llama ".gitignore" en la carpeta raíz de tu proyecto. Como puedes observar, es un archivo oculto, ya que comienza por un punto ".".

Nota: Los archivos cuyo nombre comienza en punto "." son ocultos solamente en Linux y Mac. En Windows los podrás ver perfectamente con el explorador de archivos.

Dentro del archivo .gitignore colocarás texto plano, con todas las carpetas que quieres que Git simplemente ignore, así como los archivos.

La notación es muy simple. Por ejemplo, si indicamos la línea

bower_components/

Estamos evitando que se procese en el control de versiones todo el contenido de la carpeta "bower_components".

Si colocamos la siguiente línea:

*.DS_Store

Estaremos evitando que el sistema de control de versiones procese todos los archivos acabados de .DS_Store, que son ficheros de esos que crea el sistema operativo del Mac (OS X) automáticamente.

El comando diff

El comando diff en Git se utiliza para mostrar las diferencias entre dos versiones de un archivo. Esto es muy útil cuando se está trabajando en un proyecto y se quiere ver qué cambios se han realizado entre diferentes commits o entre el directorio de trabajo y el último commit.

Supongamos que tienes un archivo llamado ejemplo.txt en tu proyecto y has realizado algunos cambios en él. Para ver las diferencias entre la versión actual del archivo y la última versión confirmada (commit), puedes ejecutar el siguiente comando:

git diff ejemplo.txt

Eliminar del repositorio archivos que ya se han confirmado (se ha hecho commit)

El problema mayor te puede llegar cuando quieras quitarte de enmedio un archivo que hayas confirmado anteriormente. O una carpeta con un conjunto de archivos que no deberían estar en el repositorio.

Una vez confirmados los cambios, es decir, una vez has hecho commit a esos archivos por primera vez, los archivos ya forman parte del repositorio y sacarlos de allí requiere algún paso adicional. Pero es posible gracias a la instrucción "rm". Veamos la operativa resumida en una serie de comandos

Borrar los archivos del repositorio

El primer paso sería eliminar esos archivos del repositorio. Para eso está el comando "rm". Sin embargo, ese comando tal cual, sin los parámetros adecuados, borrará el archivo también de nuestro directorio de trabajo, lo que es posible que no desees.

Si quieres que el archivo permanezca en tu ordenador pero simplemente que se borre del repostorio tienes que hacerlo así.

git rm --cached nombre_archivo

Si lo que deseas es borrar un directorio y todos sus contenidos, tendrás que hacer algo así:

git rm -r --cached nombre_directorio

El parámetro "--cached" es el que nos permite mantener los archivos en nuestro directorio de trabajo.

El comando git reset --hard

Se utiliza para descartar todos los cambios realizados en el directorio de trabajo y en el área de preparación (staging area), y reestablecer el repositorio a un estado anterior.

Cuando ejecutas git reset --hard, Git reestablece el directorio de trabajo, el área de preparación y la rama actual al mismo estado que el último commit especificado. Esto significa que todos los **cambios** realizados desde ese último commit se perderán de manera definitiva.

git reset --hard 12345abc

El comando git reset --mixed

El comando git reset --mixed en Git es similar a git reset --hard, pero con una diferencia importante: en lugar de descartar todos los cambios, el comando git reset --mixed mueve los cambios del área de preparación (staging area) de vuelta al directorio de trabajo.

Esto significa que los cambios realizados en los archivos todavía existen en el directorio de trabajo, pero ya no están en el área de preparación, es decir, no se incluirán en el próximo commit.

git reset --mixed 12345abc

El comando git reset --soft

El comando git reset --soft en Git es similar a git reset --mixed, pero en lugar de mover los cambios del área de preparación (staging area) al directorio de trabajo, los mantiene en el área de preparación.

Esto significa que los cambios realizados en los archivos se mantienen en el área de preparación, y no se eliminan ni se mueven al directorio de trabajo.

git reset --soft 12345abc

comandos git stash, git stash list, git stash pop y git stash drop

git stash:

- Este comando guarda temporalmente los cambios que has realizado en tu directorio de trabajo y en el área de preparación (staging area).
- Los cambios se "esconden" en una pila de cambios, sin necesidad de hacer un commit.
- Esto es útil cuando quieres cambiar de rama o realizar otras tareas, pero no quieres perder los cambios en los que has estado trabajando.

git stash list:

- Este comando muestra una lista de todos los cambios que has guardado en la pila de stash.
- Cada entrada en la lista tiene un nombre único, que te permite identificar los diferentes conjuntos de cambios que has guardado.

git stash pop:

- Este comando restaura los cambios más recientes de la pila de stash y los aplica en tu directorio de trabajo y en el área de preparación.
- Después de ejecutar este comando, los cambios se eliminan de la pila de stash.

git stash drop:

- Este comando elimina el stash más reciente de la pila de stash, sin restaurar los cambios.
- Esto es útil cuando ya no necesitas los cambios que has guardado en el stash.

En resumen, git stash te permite guardar temporalmente tus cambios sin necesidad de hacer un commit, git stash list te muestra la lista de stashes, git stash pop restaura los cambios más recientes, y git stash drop elimina un stash de la pila.

Estas herramientas son muy útiles cuando necesitas cambiar de contexto en tu trabajo sin perder los cambios en los que has estado trabajando.

Trabajar con ramas en Git: git branch

La rama master

Cuando inicializamos un proyecto con Git automáticamente nos encontramos en una rama a la que se denomina "master".

Puedes ver las rama en la que te encuentras en cada instante con el comando:
git branch

Esta rama es la principal de tu proyecto y a partir de la que podrás crear nuevas ramas cuando lo necesites.

Crear una rama nueva

El procedimiento para crear una nueva rama es bien simple. Usando el comando branch, seguido del nombre de la rama que queremos crear.

git branch experimental

Este comando en sí no produce ninguna salida, pero podrías ver las "branches" de un proyecto con el comando "git branch", u obtener una descripción más detallada de las ramas con este otro comando:

git show-branch

Esto nos muestra todas las ramas del proyecto con sus commits realizados. La salida sería como la de la siguiente imagen.

Pasar de una rama a otra

Para moverse entre ramas usamos el comando "git checkout" seguido del nombre de la rama que queremos que sea la activa.

git checkout experimental

esta sencilla operación tiene mucha potencia, porque nos cambiará automáticamente todos los archivos de nuestro proyecto, los de todas las carpetas, para que tengan el contenido en el que se encuentren en la correspondiente rama.

De momento en nuestro ejemplo las dos ramas tenían exactamente el mismo contenido, pero ahora podríamos empezar a hacer cambios en el proyecto experimental y sus correspondientes commit y entonces los archivos tendrán códigos diferentes, de modo que puedas ver que al pasar de una rama a otra hay cambios en los archivos.

El comando checkout tiene la posibilidad de permitirte crear una rama nueva y moverte a ella en un único paso. Para crear una nueva rama y situarte sobre ella tendrás que darle un nombre y usar el parámetro -b.

git checkout -b otrarama

Como salida obtendrás el mensaje Switched to a new branch 'otrarama'. Eso quiere decir que, además de crear la rama, nuestra cabecera está apuntando hacia esta nueva branch.

Borrar una rama

Borrado de la rama en local

Esto lo conseguimos con el comando git branch, solamente que ahora usamos la opción "-d" para indicar que esa rama queremos borrarla.

git branch -d rama_a_borrar

Fusionar ramas

El comando git merge

El comando git merge en Git se utiliza para fusionar los cambios de una rama con otra.

Fusionar dos ramas:

git merge <nombre_rama>
 # Por ejemplo:
git merge feature/nueva-funcionalidad

Resolver conflictos durante la fusión:

Cuando hay conflictos entre los cambios de las ramas que se van a fusionar, Git te pedirá que los resuelvas manualmente. Puedes editar los archivos con conflictos, elegir qué cambios conservar, y luego completar la fusión.

TRABAJO EN REMOTO CON GITHUB

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Algunas de las principales características y funcionalidades de GitHub son:

1. **Control de versiones**: GitHub utiliza Git como sistema de control de versiones, lo que permite a los desarrolladores trabajar de forma colaborativa en un mismo proyecto, hacer seguimiento de los cambios y revertir a versiones anteriores si es necesario.

2. **Repositorios**: GitHub permite a los usuarios crear y alojar repositorios, que son directorios donde se almacenan los archivos de un proyecto. Estos repositorios pueden ser públicos o privados, y permiten a los desarrolladores compartir y colaborar en el código.

3. **Colaboración**: GitHub facilita la colaboración entre desarrolladores. Permite que varias personas trabajen en un mismo proyecto, realicen revisiones de código, comenten y discutan sobre él, y resuelvan problemas de manera conjunta.

4. **Seguimiento de problemas**: GitHub proporciona un sistema de seguimiento de problemas (issues), donde los usuarios pueden informar de errores, solicitar nuevas funcionalidades o discutir sobre el proyecto.

5. **Ramas y fusión de código**: GitHub permite a los desarrolladores crear ramas (branches) para trabajar de manera independiente en características o correcciones, y luego fusionar (merge) estos cambios con la rama principal del proyecto.

6. **Páginas web**: GitHub ofrece la posibilidad de crear y alojar páginas web estáticas a través de GitHub Pages, lo que facilita la publicación y el alojamiento de sitios web, documentación y portafolios.

7. **Comunidad**: GitHub es una de las mayores comunidades de desarrolladores a nivel mundial. Permite a los usuarios descubrir, contribuir y aprender de otros proyectos y desarrolladores.

En resumen, GitHub es una plataforma esencial para el desarrollo de software en equipo, la colaboración, el control de versiones y el alojamiento de proyectos, que ha se ha convertido en una herramienta fundamental para la comunidad de desarrollo de software a nivel global.

El comando git remote

El comando git remote en Git se utiliza para administrar los repositorios remotos asociados con tu repositorio local. Algunos de los usos más comunes de este comando son:

Mostrar los repositorios remotos configurados:

git remote -v
Agregar un nuevo repositorio remoto:

git remote add <nombre_remoto> <url_del_repositorio>
 # Por ejemplo:
git remote add origin https://github.com/usuario/proyecto.git

Cambiar la URL de un repositorio remoto:

git remote set-url <nombre_remoto> <nueva_url>
 # Por ejemplo:
git remote set-url origin https://github.com/usuario/nuevo-proyecto.git

Eliminar un repositorio remoto:

git remote remove <nombre_remoto>
 # Por ejemplo:
git remote remove origin

Mostrar información detallada sobre un repositorio remoto:

git remote show <nombre_remoto>
 # Por ejemplo:
git remote show origin

El comando git push

El comando git push en Git se utiliza para enviar (publicar) los cambios de tu repositorio local a un repositorio remoto.

git push <nombre_remoto> <nombre_rama>
 # Por ejemplo:
git push origin master

Subir una nueva rama al repositorio remoto:

git push <nombre_remoto> <nombre_rama_local>:<nombre_rama_remota>
 # Por ejemplo:
git push origin feature/nueva-funcionalidad:feature/nueva-funcionalidad

Ejemplo

crear un nuevo repositorio en la línea de comando

echo "# prueba3" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/Ron3333/prueba3.git
git push -u origin master

enviar un repositorio existente desde la línea de comando

git remote add origin https://github.com/Ron3333/prueba3.git
git branch -M master
git push -u origin master

El comando git fetch

El comando git fetch en Git se utiliza para descargar objetos y referencias de un repositorio remoto.

git fetch <nombre_remoto>
 # Por ejemplo:
git fetch origin

Este comando descargará todos los objetos (commits, ramas, etiquetas, etc.) del repositorio remoto llamado "origin", pero no fusionará (merge) los cambios con tu repositorio local.

Descargar cambios de una rama remota específica:

git fetch <nombre_remoto> <nombre_rama_remota>
 # Por ejemplo:
git fetch origin feature/nueva-funcionalidad

El comando git pull

El comando git pull en Git se utiliza para descargar contenido de un repositorio remoto y actualizar el repositorio local inmediatamente.

git pull <nombre_remoto> <nombre_rama>
 # Por ejemplo:
git pull origin master

Este comando descargará los cambios de la rama "master" del repositorio remoto "origin" y los fusionará (merge) automáticamente con tu rama local.

Descargar y fusionar cambios sin especificar la rama remota:

git pull

Este comando descargará los cambios de la rama remota que esté configurada como la rama de seguimiento de tu rama local actual, y los fusionará con tu rama local.

El comando git clone

El comando git clone en Git se utiliza para crear una copia de un repositorio remoto en tu sistema local.

git clone <url_del_repositorio_remoto>
 # Por ejemplo:
git clone https://github.com/usuario/mi-proyecto.git

Este comando descargará una copia completa del repositorio remoto en tu sistema local, incluyendo toda la historia y las ramas del proyecto.

NOTAS

Comando SSH: ssh-keygen -t rsa -C "nombre-correo@gmail.com"

Video recomendado de curso de git

https://www.youtube.com/watch?v=3GymExBkKjE