# **Aprender a usar el terminal GIT con ejemplos**

Si sueles trabajar con alguna interfaz gráfica y quieres aprender los conceptos básicos para trabar usando el terminal, continúa leyendo.

En este Readme encontrarás paso a paso un tutorial guiado que te ayudará a aprender de forma practica como utilizar el terminal.

En este repositorio tienes un proyecto básico (un proyecto web que muestra un número por la consola), y durante el desarrollo del mismo te proporcionaremos una guía para:

* Recrear el repositorio desde 0.
* Realizar puntos de confirmación (Commit a partir de ahora) y subir los cambios (de ahora en adelante lo llamaremos Push).
* Crear ramas.
* Unir ramas.
* Resolver conflictos.

Usaremos un terminal de comando git para configurar y trabajar con el repositorio.

Prerequisitos para realizar este tutorial:

* Instalar [Git](https://git-scm.com/downloads).
* Instalar [nodejs](https://nodejs.org/es/).

# **Configuración de usuario**

Es importante antes de comenzar establecer tu nombre de usuario y dirección de correo electrónico, ya que en los "commits" que realicemos se guardará dicha información.

1. Configurar tu nombre de usuario en Git

$ git config --global user.name "Nombre prueba"

1. Configurar tu dirección de correo electrónico

$ git config --global user.email prueba@ejemplo.com

Si quieres comprobar tu configuración, puedes usar el comando:

$ git config --list

# **Creación del repositorio**

Se puede elegir entre usar una conexión HTTPS (con tu usuario estándar de git) o una conexión SSH (una aproximación más seguro para tratar con tus repositorios, pero necesita una configuración adicional)

Mas información sobre como [crear una clave SSH en tu ordenador](https://confluence.atlassian.com/bitbucketserver/creating-ssh-keys-776639788.html), y [configurarlo en Git](https://help.github.com/en/enterprise/2.18/user/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account)

Para crear el repositorio solo tienes que pulsar en el botón nuevo repositorio en Github o en tu proveedor de repositorios favorito.

Cuando creas un nuevo proyecto, es una buena práctica seleccionar la opción de crear un fichero readme.md por defecto. Realizando este paso, tendremos nuestro commit inicial.

# **Clonar repositorio**

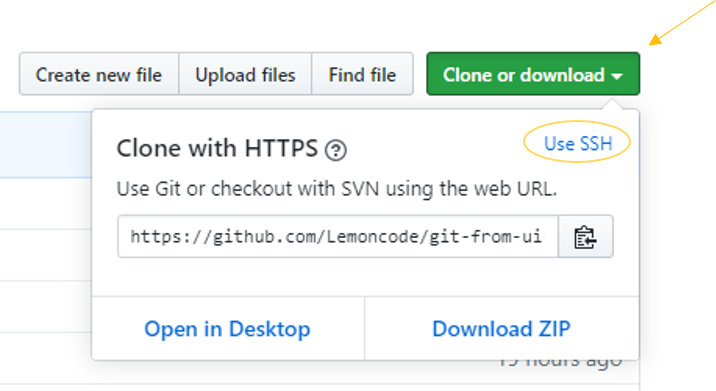
Ahora que ya tienes tu repositorio creado, es hora de traerlo a tu equipo de forma local, para hacer eso lo clonamos (descargar el repositorio a una carpeta local del ordenador).

Asegúrate que tienes permiso para acceder a esos repositorios, y si estas usando SSH tienes que tenerlo correctamente configurado (generada localmente tu clave privada y publica SSH, y subida tu clave publica a tu repositorio).

Elige una carpeta para descargar los ficheros, abriendo un terminal windows y ejecutando el siguiente comando:

git clone <repository\_address>

Normalmente los proveedores de repositorios git te muestran en algún cuadro de texto la dirección url / ssh del repositorios para que la puedas copiar.

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/clone-with-ssh-or-https.png)

# **Añadir un proyecto web básico**

\*\* Esto no está relacionado con Git, sólo tenemos que crear una proyecto básico para jugar con el\*\*

Vamos a añadirle contenido al repositorio.

Empezaremos creando un proyecto web básico (configuramos nodejs y un proyecto node que muestra un mensaje por consola), vamos a ejecutar el siguiente comando:

npm init -y

e instalamos un bundler:

npm install parcel rimraf --save-dev

Vamos a crear el siguiente fichero en la carpeta src:

./src/index.js

const sampleNumber = 1;console.log(`Hello number ${sampleNumber}`);

Vamos a crear una pagina HTML para el inicio de la aplicación.

./src/index.html

<html>

<body>

<h1>Check the console log</h1>

<script src="./index.js"></script>

</body></html>

Vamos a añadir el siguiente comando al fichero package.json:

./package.json

"scripts": {+ "start": "rimraf dist && parcel ./src/index.html --open",

"test": "echo \"Error: no test specified\" && exit 1"

},

Es hora de comprobar si nuestro proyecto funciona, en el termina ejecutamos:

npm start

Se abrirá una ventana del navegador y si tienes abierta la consola (por ejemplo en windows pulsando F12) puedes comprobar si se muestra el mensaje.

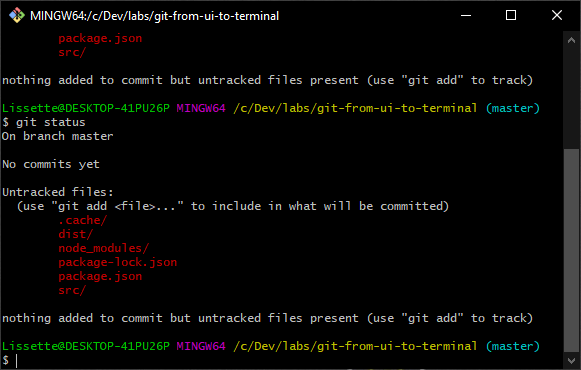
Hasta ahora todo bien... ya tenemos un proyecto basico para jugar con el, comencemos a trabajar con Git!

# **Haciendo nuestro primer commit**

Quizás pensemos que ahora estamos preparado para hacer nuestro primer commit en la rama master (es la rama por defecto se crea en nuestro proyecto)

Antes de hacerlo, chequeamos que ficheros han sido modificado, para esto abrimos el termina y escribimos el siguiente comando:

git status

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/initial-git-status.PNG)

Como ???? Sólo hemos creado unos pocos ficheros... ¿Por qué tenemos esa enorme cantidad de ficheros en nuestra lista? Bien npm install los creó, puedes ver que en la carpeta \_node\_modules están todos los ficheros que son necesario para ejecutar la aplicación pero no son necesario en el repositorio (y no debes subirlos), tenemos que ignorarlos, para ello creamos un fichero llamado \_.gitignore\_ e incluimos el siguiente contenido:

./.gitignore

/node\_modules

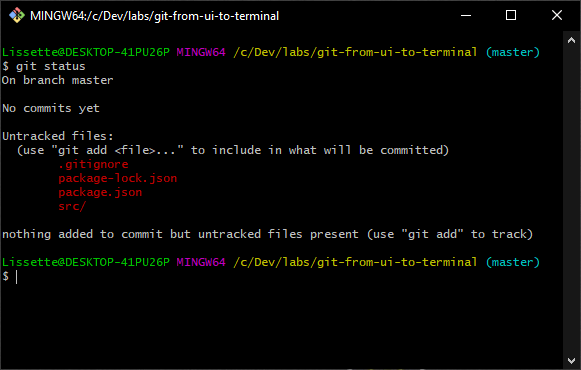
/dist

/.cache

Haciendo esto, definimos que se ignore todos los ficheros incluidos en la carpeta node\_modules (no se subirá a nuestro repositorio), [pulsa aquí para obtener mas información sobre como incluir entradas y patrones en un fichero .gitIgnore](https://git-scm.com/docs/gitignore).

Ahora si volvemos a ejecutar el comando status podemos ver que solo se muestran los ficheros que hemos creado y están pendientes de ser añadido al repositorio:

git status

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/after-gitignore-git-status.PNG)

Ahora si queremos subir estos cambios al nuestro servidor git, necesitamos realizar 3 pasos:

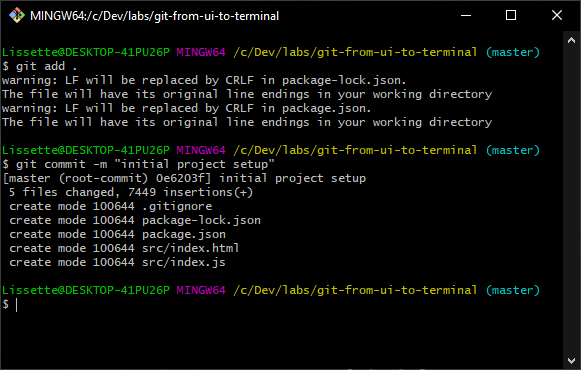
* Primero, necesitas marcar esos ficheros como listos para ser incluidos en el commit (puedes hacerlo fichero por fichero o seleccionando todos los ficheros).
* Segundo, necesitas hacer el commit de los ficheros en tu base de datos local de git.
* Tercero, ahora puedes hacer push de estos cambios al servidor git.

Vamos a marcar a los ficheros como candidatos al commit (estado staged)

git add .

Ahora podemos hacer commit a nuestra base de datos local, añadiremos el parámetro -m para añadir un mensaje al commit:

git commit -m "initial project setup"

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-commit.PNG)

Ya estamos listos para subir los cambios locales al servidor:

git push

Para comprobar que todo ha ido bien, abre tu navegador, abre la web de tu proveedor git y comprueba que el contenido subido está disponible.

# **Crear y unificar ramas**

Ahora que hemos realizado los primeros pasos usando git es hora de avanzar, vamos a aprender los comando básicos para la creación de ramas.

Una rama nos permite crear una copia aislada de la rama master (o cualquier otra rama) y trabajar aisladamente del resto del equipo. Esto es util para:

* Evitar añadir funcionalidades no finalizadas a nuetro producto.
* Minimizar el impacto que produce a los miembros del equipo de desarrollo cuando se añaden cambios realizados por otros compañeros.
* Revisar el trabajo realizado antes de incluirlo en la rama master.

Vamos a implementar una nueva funcionalidad en nuestra aplicación: mostrar en la consola de log un segundo numero. Para evitar el impacto a los otros miembros del equipo mientras desarrollamos nuestra mejora, vamos a crear una rama. Para ello ejecutamos el siguiente comando:

git branch feature/display-number-b

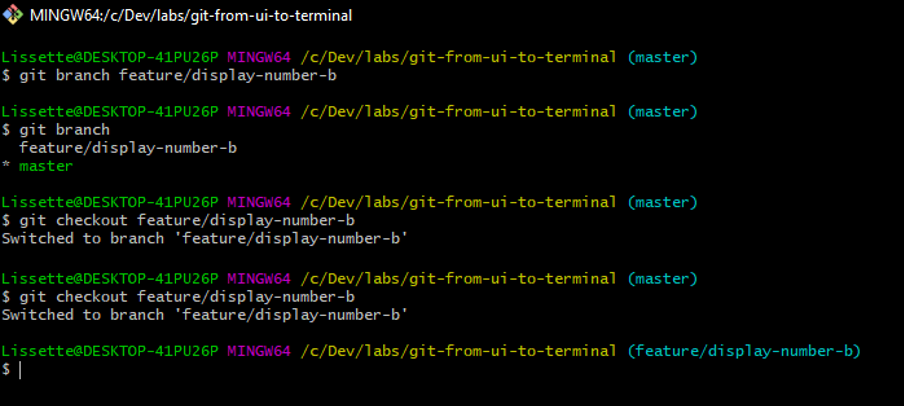
Sobre la convención de nombres para la rama creada, estamos usando gitflow [pulsa aquí para obtener mas información sobre como nombrar las ramas y el uso de gitflow](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

Podemos ejecutar el comando git branch para comprobar que la rama ha sido creada correctamente. Este comando nos indicará la rama activa (master):

git branch

Vamos a cambiar de nuestra rama master a la rama feature/display-number-b

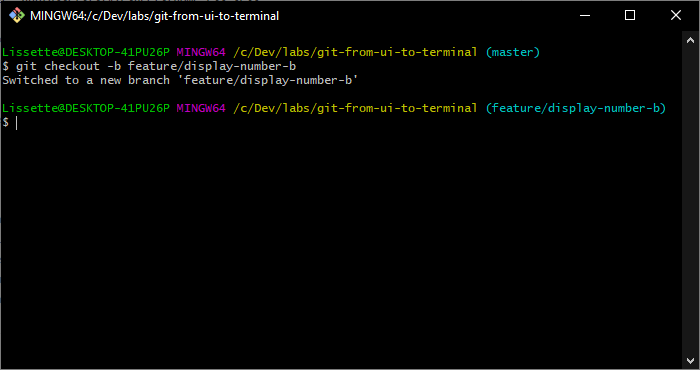
git checkout feature/display-number-b

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-checkout-branch.png)

Puedes cambiar de una rama a otra si la rama activa no tiene cambios pendientes (Si la rama activa tiene cambios pendientes, tienes que hacer un commit de ellos, descartarlos o almacenarlos en un espacio temporal, veremos esto mas adelante en el tutorial).

Otra fomar más rapida es crear una rama y cambirse a ella directamente ejecutando este comando:

git checkout -b feature/display-number-b

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-checkout-b.PNG)

Ahora podemos empezar a trabajar en nuestra rama sin provocar ningún impacto en los otros desarrolladores que estan trabajando en la rama master.

Vamos a añadir algunos cambios al fichero .src/index.js:

./src/index.js

const sampleNumber = 1;+ const sampleNumberB = 2;- console.log(`Hello number ${sampleNumber}`);+ console.log(`Hello number ${sampleNumber} {sampleNumberB}`);

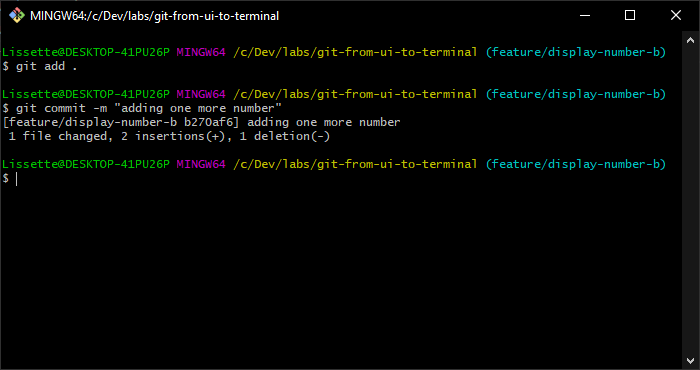
Guardamos los cambios, y los añadimos al commit:

git add .

Este comando añade al commit cualquier fichero nuevo o modificado.

Ya podemos hacer commit de los ficheros en nuestro base de datos local:

git commit -m "adding one more number"

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-commit-first-change.PNG)

Y subimos los cambios (push) a nuestro servidor remoto (En esta caso la rama nueva no existe en el servido, necesitamos añadir un parametro para indicar que se va a crear la rama en el servidor origin)

git push -u origin feature/display-number-b

Ahora podriamos volcar (merge) nuestra rama a la rama de master, pero no lo vamos a hacer por ahora: primero vamos a simular que otro desarrollador crea otra rama y modifica el fichero ./src/index.js, esto provocará un conflicto, que aprenderemos a resolver en la siguiente sección del tutorial.

Un buen consejo: nunca hagas un merge directamente de una rama a master, en vez de eso, realiza una petición para que revisen tu código (puedes realizar esta petición usando el cliente web de tu repositorio preferido de git), por otro lado es una buena práctica mergear primero la rama master a tu rama usando el terminal, de esta forma actualizaras tu código con los cambios que se han realizado en master.

# **Deshacer un commit local**

Explicaremos una de las técnicas para deshacer un determinado commit.

1. Ubicados en la rama master o en la que estemos trabajando, buscamos en el historial para localizar el commit a eliminar:

git log --oneline

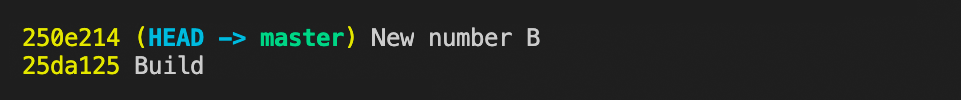
En nuestro caso optenemos lo siguiente:

[](https://github.com/Lemoncode/git-from-ui-to-terminal/blob/master/content/git-log.png)

1. Si quisieramos volver al estado de la confirmación 250e214...

git reset --hard 250e214

Ahora, si volvemos a ejecutar git log --oneline nuestro historial se muestra así...

[](https://github.com/Lemoncode/git-from-ui-to-terminal/blob/master/content/git-reset-hard.png)

Comprobamos que hemos ****borrado todo el historial**** posterior a 250e214, por lo que si ahora realizamos ****git push****, Git interpretaría que hay un error en la rama ya que faltarían confirmaciones. Por este motivo, ****git reset**** es más indicado para trabajo en local o rama privada.

****git reset**** dispone de varias ****opciones o flags**** para decidir a cuál o cuáles de los tres estados o árboles de Git afectamos, veamos:

1. ****--soft****

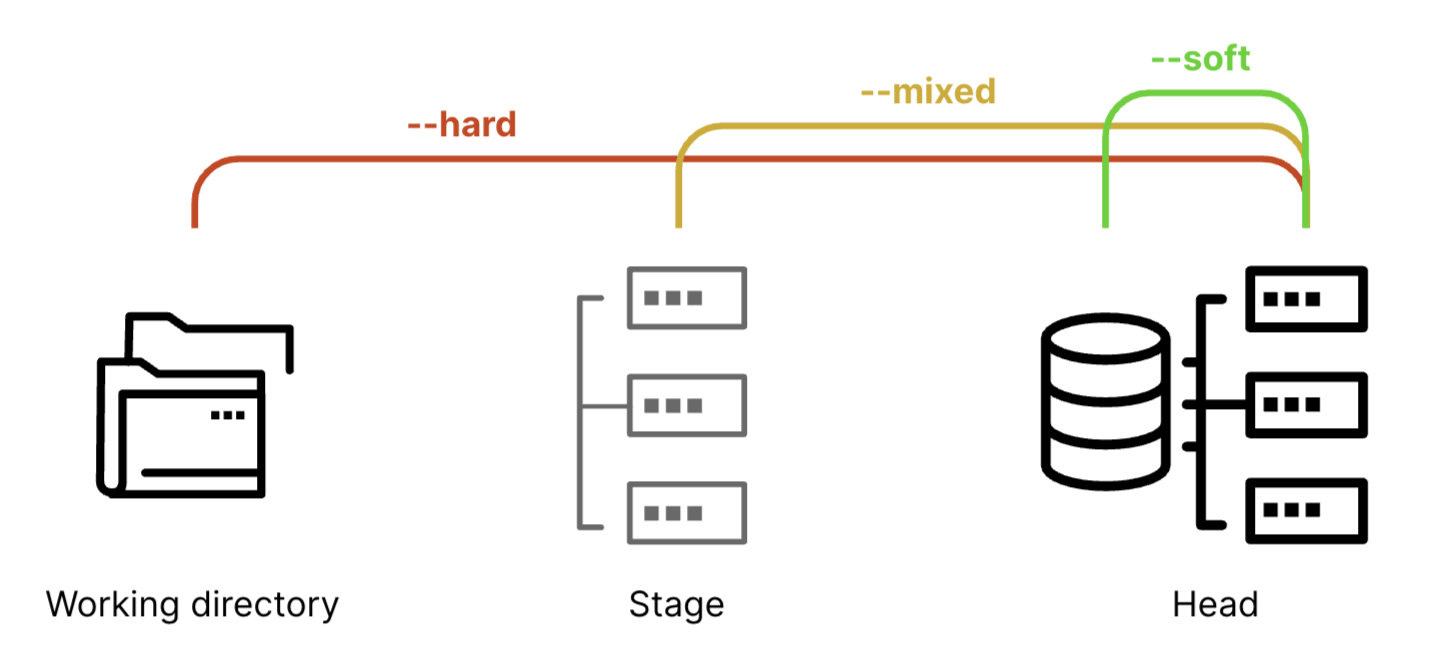
Es la opción menos destructiva, solo elimina el commit, únicamente afecta a HEAD (historial de confirmaciones) y se detiene ahí, a la espera del commit.

1. ****--mixed****

Es el comportamiento predeterminado, lo mismo que ejecutar git reset. Elimina tanto las confirmaciones como el contenido del Stage, nos quedamos al aespera añadir modificaciones con ****git add .****

1. ****--hard****

El visto en el ejemplo y el más destructivo, se eliminan: confirmaciones, estado del Stage y modificaciones en el directorio de trabajo. De hecho, si ejecutamos git status después de un reset --hard optendremos un repositorio limpio.

[](https://github.com/Lemoncode/git-from-ui-to-terminal/blob/master/content/git-reset.png)

# **Deshacer un commit público**

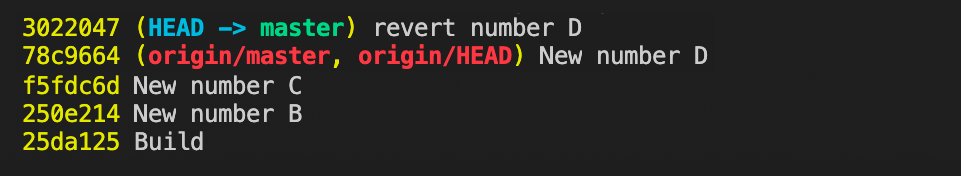
Otro modo de deshacer una confirmación es usar ****git revert****. Este es un modo más indicado cuando trabajamos con repositorios públicos o con un equipo, ya que a diferencia de reset, no eliminará el historial o no desplazará el puntero, sino que creará un nuevo commit, o mejor dicho; esperárá ha que creemos un nuevo commit.

Si lo ejecutamos, el archivo quedará modificado justo al estado anterior (volvemos un paso atrás) y por tanto será necesario "comitearlo"

Ejecutamos revert en un punto similar al anterior...

git revert HEAD

realizamos el nuevo commit con las modificaciones y revisamos el historial con git log...

[](https://github.com/Lemoncode/git-from-ui-to-terminal/blob/master/content/git-revert.png)

Vemos que las confirmaciones se mantienen, por lo que Git no interpretará un error al realizar ****git push.****

# **Manejando conflictos**

Antes de realizar el volcado de nuestra nueva rama a master, vamos a simular que otro programador (o nosotros mismos) ha creado una segunda rama y modifica los ficheros que nosotros acabamos de actualizar.

Cambiemos a la rama de master:

git checkout master

Vamos a crear una nueva rama partiendo de master:

git branch feature/display-number-c

Cambiemos a esa nueva rama:

git checkout feature/display-number-c

Y modificamos el fichero index.js y incluimos un numberC para mostrarlo

/.src/index.js

const sampleNumber = 1;+ const sampleNumberC = 3;- console.log(`Hello number ${sampleNumber}`);+ console.log(`Hello number ${sampleNumber} ${sampleNumberC}`);

Añadimos al commit los ficheros modificados:

git add .

Vamos a hacer commit de este fichero.

git commit -m "Adding third number"

Y subimos los cambios:

git push -u origin feature/display-number-c

Comprobemos las listas de ramas que tenemos disponibles:

git branch

Tenemos tres ramas: master, feature/display-number-b, feature/display-number-c y la rama activa feature/display-number-c

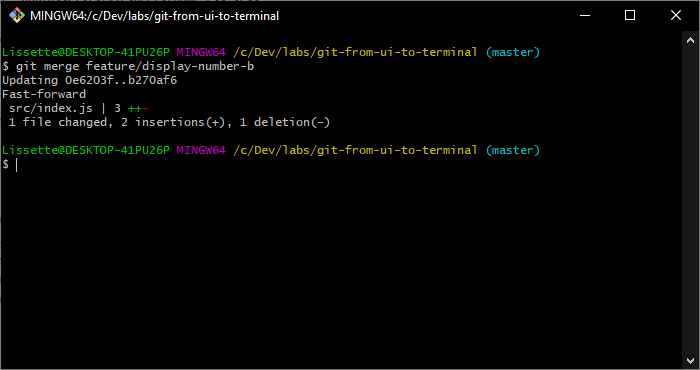
Suponemos que tenemos el OK de nuestro equipo para mergear la rama feature-display-number-b a master, vamos a ellos.

Cambiemos a la rama master

git checkout master

Mergeamos featute/display-number-b en master:

git merge feature/display-number-b

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-merge-branch-display-number-b.PNG)

Una buena practica antes de mergear una rama a master es asegurarse que la rama está actualizada con la última versión de master (mergear master en nuestra rama)

Esto lo podemos hacer porque tenemos las ramas disponibles localmente, si es una rama creada por otro usuario puede ser que solo este disponible en el servidor y no en mi equipo local. Tendriamos que ejecutar el comando fetch para traer dicha rama a nuestro equipo (ver en la sección de varios).

En este caso no hay conflictos, por tanto podemos continuar.

Vamos a subir los cambios al servidor.

git push

Ahora puedes usar tu navegador web e ir a la pagina de tu proveedor de respositorios y comprobar que se han subido los cambios, genial!

El siguiente paso será mas complejo... queremos mergear la rama feature/display-number-c en master, ¿Por qué es más dificil en esta ocasión? Porque la version de master de la que partimos para crear la rama feature/display-number-c es diferente a la actual, y hay conflictos (el ficheroindex.js es diferentes en ambas versiones). Necesitamos resolver los conflictos y seleccionar que parte de codigo es la correcta para el buen funcionamiento de la aplicación.

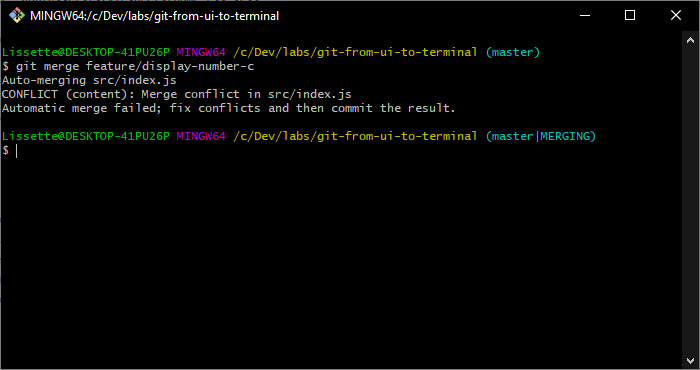
Vamos a mergear feature/display-number-c en master.

Asegurate primero que estas en la rama de master, ejecutando el comando git branch obtendremos la lista de ramas disponibles y se verá resaltada la rama activa.

git branch

Ahora volcamos esta rama en master.

git merge feature/display-number-c

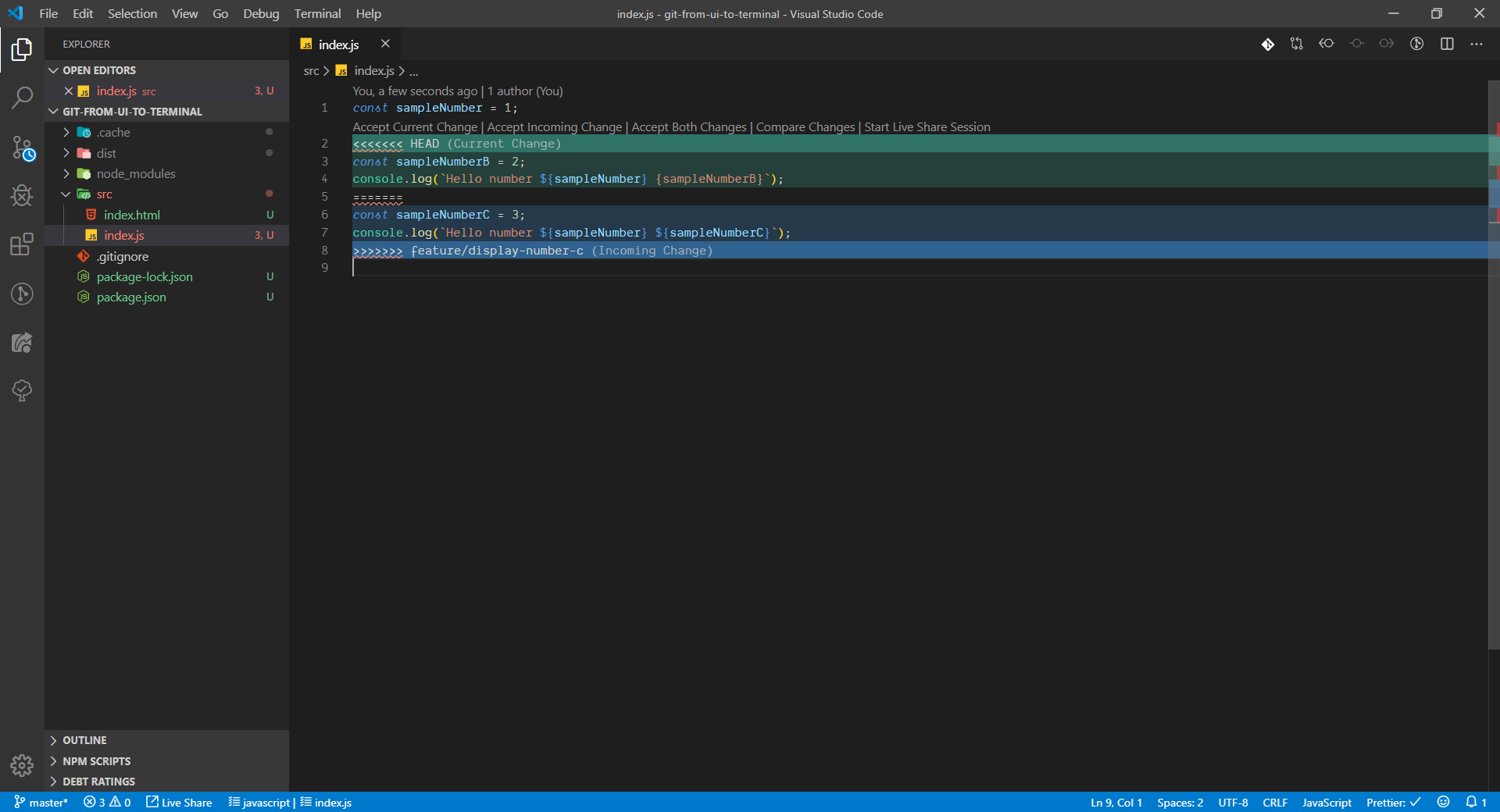
[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-merge-branch-display-number-c-with-conflicts.PNG)

****Ops tenemos conflictos !!**** Tenemos que resolverlo con el comando git mergetool.

Antes de lanzar este comando, vamos a realizar la siguiente comprobación, tienes alguna herramienta de resolución de conflictos instalada y configurada? Si no la tienes, salta a la seccion ****otros/Configurar una herramienta de resolución de conflictos**** de este tutorial.

Vamos a lanzar el comando para empezar el merge:

git mergetool

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/vscode-mergetool.PNG)

Dependiendo de tu configuración, se lanzará una herramienta (KDiff, VSCode, p4Merge...). Ahora tiene que ir recorriendo todos los conflictos y eligiendo (normalmente tienes que elegir conflicto por conflicto):

* Conservar tus cambios locales y descartar los cambios de la rama donde se ha mezclado los cambios.
* Aplicar los cambios que se han producido en la mezcla de ambas ramas.
* Modificar tu mismo el codigo para resolver el conflicto de forma manual.

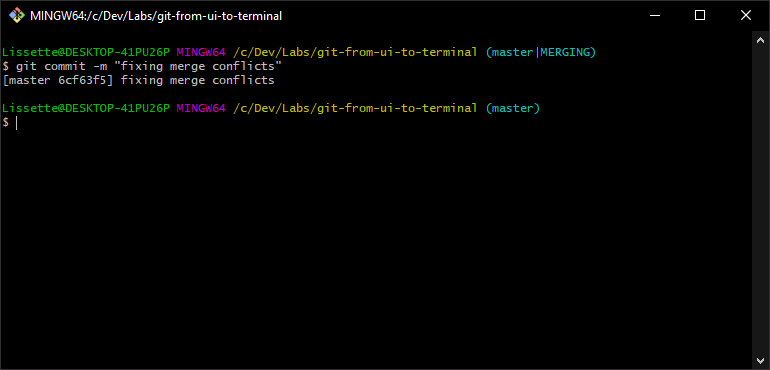
Dependiendo de la herramienta, se intentará resolver de forma automática algunos conflictos.

Si no tienes configurada una herramienta de resolución de conflictos, o quieres aprender mas sobre ella, consulta la seccion Otros/Configurar una herramienta de resolución de conflictos

Una vez que tengas todos los conflictos resueltos, pasamos los ficheros cambiados a staging y tendras que hacer un commit.

git add .

git commit -m "fixing merge conflicts"

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-commit-merge-conflicts.PNG)

Algunos desarrolladores prefieren usar herramientas con interfaz gráfica para manejar estos casos (por ejemplo Source Tree, Git Kraken, VSCode).

Ahora podemos hacer push para subir los cambios al servidor.

git push

Tienes muchos ficheros temporales con la extensión \*.orig? Puedes configurar la instalación de git para que no guarde esos ficheros:

git config --global mergetool.keepBackup false

# **Otros**

## **Crear un repositorio en una máquina local y subirlo después a Github**

Creamos en local la carpeta de nuestro directorio o utilizamos la carpeta de un proyecto ya existente que queramos subir al repositorio de Github.

Accedemos a dicha carpeta con la terminal y escribimos el siguiente comando para crear la base de datos local (si la base de datos ya existe en dicha ubicación se reinicializará):

git init

1. A continuación, es necesario vincular la base de datos local con Github. Para ello tenemos que acceder a la página web y crear un repositorio completamente vacío (sin el archivo README inicial). Copiamos la url del repositorio y escribimos el siguiente comando:

git remote add origin https://github.com/...

1. En este punto ya tendríamos la base de datos local y el repositorio de Github vinculados, por lo que sólo sería necesario subir los ficheros a dicha plataforma. Para ello empezamos alojando dichos ficheros en el staging:

git add .

1. Posteriormente guardamos los ficheros en la base de datos local:

git commit -m "comentario"

1. Por último, ya podríamos subir los archivos a Github. En este caso no bastaría con hacer un push, ya que tendríamos que setear la rama master del repositorio:

git push --set-upstream origin master

## **Configurar una herramienta para hacer los merges**

Existe multitud de herramientas para hacer los merges, y decidir cual utilizar puede ser dificil, por ejemplo:

* Kdiff
* VSCode
* p4Merge
* ...

En este [post](https://www.git-tower.com/blog/diff-tools-windows/) puedes ver las disponibles para Windows, en este [otro](https://www.tecmint.com/best-linux-file-diff-tools-comparison/) para Linux, y para Mac teneis este [post](https://www.lawtechnologytoday.org/2017/11/mac-comparing/).

[Como configurar una herramienta para hacer los merges en Mac](https://coderwall.com/p/3wuuda/set-diffmerge-as-default-merge-tool-in-os-x)

Por ejemplo:

Configurar Kdiff en windows:

* Instalar Kdiff, puedes decargarlo [aquí](https://sourceforge.net/projects/kdiff3/files/latest/download)
* Abrir el terminal y ejecutar los siguientes comandos:

git config --global --add merge.tool kdiff3&&

git config --global --add mergetool.kdiff3.path "C:/Program Files/KDiff3/kdiff3.exe"&&

git config --global --add mergetool.kdiff3.trustExitCode false&&

git config --global --add diff.guitool kdiff3&&

git config --global --add difftool.kdiff3.path "C:/Program Files/KDiff3/kdiff3.exe"&&

git config --global --add difftool.kdiff3.trustExitCode false

Configurar Kdiff en Mac:

* Instalar Kdiff, puedes descargarlo [aquí](https://sourceforge.net/projects/kdiff3/files/latest/download)
* Abrir el terminal y ejecutar los siguientes comandos:

git config --global --add merge.tool kdiff3&&

git config --global --add mergetool.kdiff3.path "Applications/kdiff3.app/Contents/MacOS/kdiff3"&&

git config --global --add mergetool.kdiff3.trustExitCode false&&

git config --global --add diff.guitool kdiff3&&

git config --global --add difftool.kdiff3.path "Applications/kdiff3.app/Contents/MacOS/kdiff3"&&

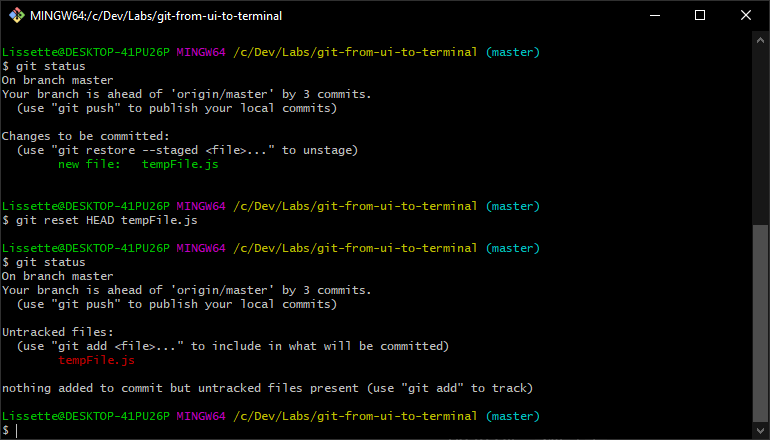
git config --global --add difftool.kdiff3.trustExitCode false

## **Jugando con el estado staging**

Cuando trabajamos localmente en nuestro repositorio, si quieres hacer commit a tus cambios, necesitas realizar los siguientes pasos: Primero añadir los ficheros al estado staging y luego hacer commit a esos ficheros.

¿Qué puedes hacer si has añadido a estado staging ficheros temporales? Lo puedes eliminar de este estado ejecutando el siguiente comando:

git reset HEAD <filepath>

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-staging.PNG)

Y... ¿si quiero eliminar todos los ficheros que estan en estado staging?

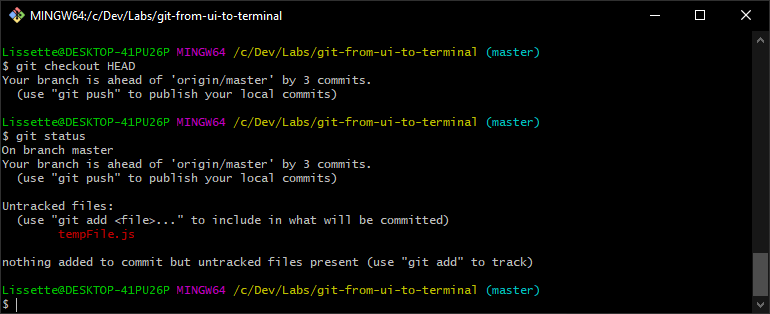
git reset HEAD .

Genial, pero ¿y si quiero descartar los cambios en un fichero que no esta en estado staging?

git checkout --

Y lo último pero no menos importante... descartar todos los cambios. Me gustaría empezar desde cero partiendo del commit anterior:

git checkout HEAD

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-checkout-head.PNG)

## **Fetch**

Cuando trabajamos en un equipor, puede que existan ramas que han sido subidas al servidor y no estan disponibles en nuestro respositorio local, ¿cómo puedo descargarmelas? Ejecuta:

git fetch --all

De vez en cuando, hay ramas que no existen en el servidor remoto pero las sigues reflejando localmente, ¿cómo puedes hacer limpieza y eliminar esas referencias?, para hacer eso, ejecutamos:

git fetch --prune

## **Pull**

En ocasiones tenemos a más de una persona trabajando en la misma rama (o por ejemplo máster puede tener cambios de otros compañeros), ¿ Qué pasa si necesita actualizar mi rama activa porque hay cambios en el servidor? Si intentamos actualizarla usando ****fetch**** puede que no funcione porque podemos tener conflictos en la rama activa, ¿No existe un comando que se traiga los cambios de la rama activa e intente mezclarlos? Si, ese comando se llama ****pull****.

git pull

## **Stash**

Algunas veces estas trabajando en tu codigo y no quieres hacer commit de los cambios, pero necesitas cambiar a otra rama, pero... para cambiar tienes que o hacer commit o descartar los cambios pendientes. ¿Hay alguna manera de poner los cambios "en borrador" y una vez que vuelvas recuperarlos? Eso es exactamente lo que hace el comando ****stash****.

Vamos a crear una nueva rama y la llamamos feature/saygoodbye

git branch feature/saygoodbye

Cambiamos a esa rama

git checkout feature/saygoodbye

Modificamos algo en el código:

./src/index.js

const sampleNumber = 1;

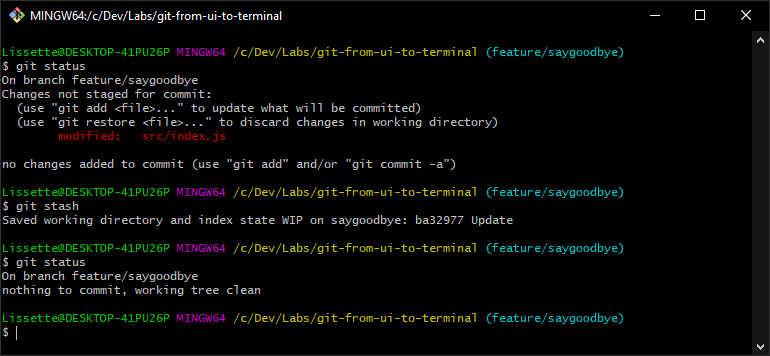
const sampleNumberB = 2;

const sampleNumberC = 3;

console.log(`Hello number ${sampleNumber} {sampleNumberB} {sampleNumberC}`);+ console.log('Goodbye !');

* No queremos hacer commit todavía, pero necesitamos cambiar a la rama master... ¿Cómo lo hacemos? Hacemos stash de nuestros cambios (todos los cambios serán almacenados localmente):

git stash

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-stash.PNG)

* Ahora podemos cambiar a la rama master

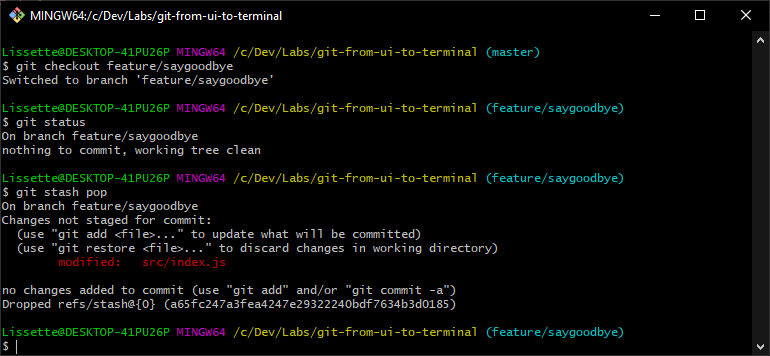
git checkout master

* Volvemos a la rama feature/saygoodbye.

git checkout feature/saygoobdye

* Y recuperamos el código que hemos almacenado en stash

git stash pop

[](https://github.com/Lemoncode/git-from-ui-to-terminal/raw/master/content/git-stash-pop.PNG)

El comando stash almacena esta información localmente y se asocia al commit específico donde ha sido llamado.

## **Migrar un repositorio a otro servidor**

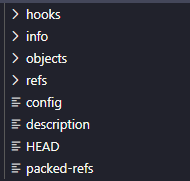
A veces, necesitamos migrar un repositorio a otro servidor y queremos que se migre todo, incluyendo ramas, referencias, tags, etc.

En vez de usar un clone normal, podemos seguir los siguientes pasos:

* Clonamos el repositorio original otra vez, pero esta vez incluyendo el flag mirror:

git clone --mirror <origin url>

* Nos habrá creado una nueva carpeta con el nombre-repositorio.git y clonará todo lo relacionado con el repositorio, algo como ésto:

[](https://github.com/Lemoncode/git-from-ui-to-terminal/blob/master/content/git-mirror.png)

* Actualizamos el terminal al nuevo directorio:

cd <cloned folder>

* Y hacemos un push a la nueva url usando el flag mirror:

git push --mirror <destination url>

## **Otros trucos**

A continuación se muestran algunas paginas donde podeis encontrar algunas trucos de git:

* <https://i.redd.it/8341g68g1v7y.png>
* <https://www.git-tower.com/blog/git-cheat-sheet/>
* <https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>

# **Sobre Basefactor + Lemoncode**

Somos un equipo innovador de expertos en Javascript, apasionados por convertir sus ideas en productos robustos.

[Basefactor, consultancy by Lemoncode](http://www.basefactor.com/) proveedor de consultoria y servicios de coaching.

[Lemoncode](http://lemoncode.net/services/en/" \l "en-home) proveedor de formación.

Disponemos de un master (castellano) en Front End, para mas información: <http://lemoncode.net/master-frontend>