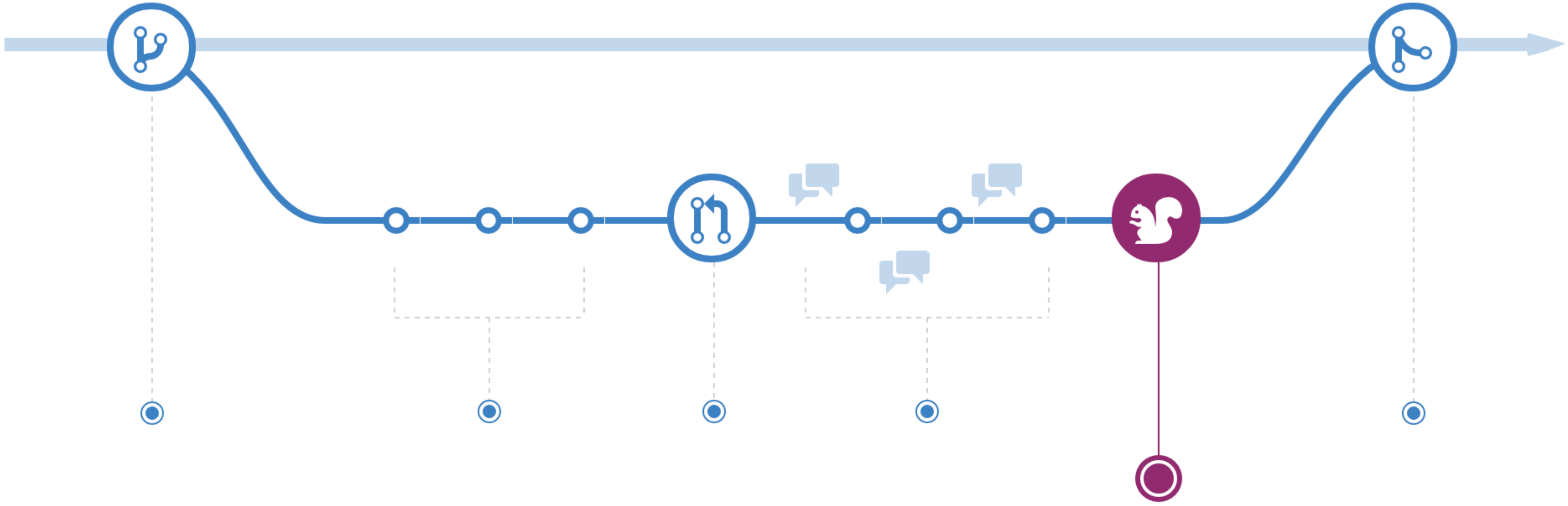
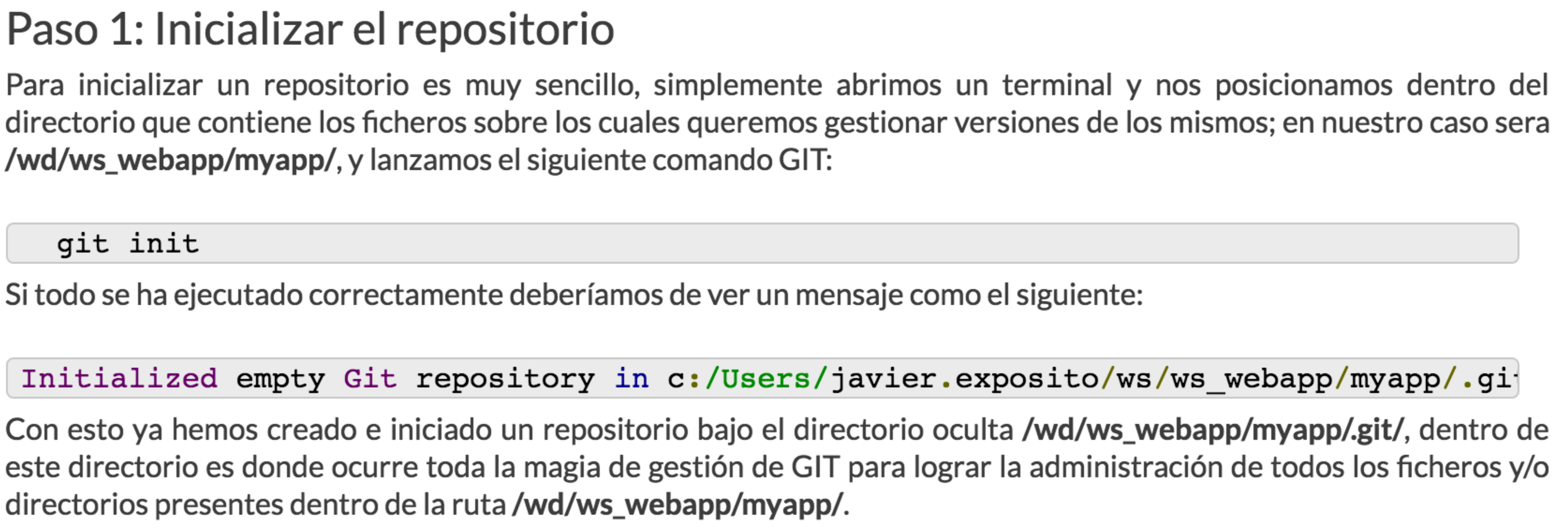


Manos a la Obra

Términos empleados

- Un **repositorio** se utiliza generalmente para organizar un solo proyecto. Los repositorios pueden contener carpetas y archivos ,imágenes ,vídeos ,hojas de cálculo y bases de datos.
- **Branching** es la manera de trabajar en diferentes versiones de un repositorio a la vez.
  En Git, los cambios realizados se llaman ***commits***. Cada commit tiene asociado un mensaje, que
- es una descripción que explica la razón de n cambio en particular.
  Pull, significa descargar los últimos cambios en el repositorio remoto y sincronizarlos con un branch local **Push**, significa hacer efectivos los commits locales y sincronizarlos con el branch remoto master

- https://guides.github.com/introduction/flow/
- https://try.github.io/levels/1/challenges/1



Paso 1: Inicializar el repositorio

Para inicializar un repositorio es muy sencillo, simplemente abrimos un terminal y nos posicionamos dentro del directorio que contiene los ficheros sobre los cuales queremos gestionar versiones de los mismos; en nuestro caso sera **/wd/ws_webapp/myapp/**, y lanzamos el siguiente comando GIT:

```
  git init
```

Si todo se ha ejecutado correctamente deberíamos de ver un mensaje como el siguiente:

```
Initialized empty Git repository in c:/Users/javier.exposito/ws/ws_webapp/myapp/.gi
```

Con esto ya hemos creado e iniciado un repositorio bajo el directorio oculta **/wd/ws_webapp/myapp/.git/**, dentro de este directorio es donde ocurre toda la magia de gestión de GIT para lograr la administración de todos los ficheros y/o directorios presentes dentro de la ruta **/wd/ws_webapp/myapp/**.

Paso 2: Determinar el estado de un repositorio

Para determinar el estado de un repositorio en GIt empleamos el comando:

```
git status
```

Este comando arroja el siguiente mensaje el cual nos indica que no existen cambios en la rama **master** del repositorio que necesiten un commit:

```
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

En GIT es posible manejar como mínimo 1 la cual corresponde con la rama **master**, pero también podemos definir o crear nuevas ramas; definimos una rama cada vez que queremos agregar funcionalidades especificas a nuestras aplicaciones; por ejemplo, una rama puede definirse para gestionar los cambios que se necesiten para una versión en especifico de la aplicación que estemos desarrollando, mas adelante hablaremos mas en detalle sobre las ramas en GIT.

Ahora vamos a crear un fichero denominado **index.html**, en el cual contedrá un template básico HTML; este fichero nlo vamos a crear dentro de la ruta en que estamos trabajando **/wd/ws_webapp/myapp/**; una vez creado volvemos a lanzar el comando **git status** y deberíamos de ver algo como lo siguiente:

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       index.html
nothing added to commit but untracked files present (use "git add" to track)
```

Ahora, el repositorio tiene un fichero denominado index.html, el cual no esta siendo monitoreado por GIT. Para poder monitorear los cambios realizados a los ficheros o directorios en necesario agregar estos al repositorio lo cual veremos a continuación.

Paso 3: Agregar cambios al repositorio

Para agregar ficheros sobre los cuales monitorear los cambios debeis emplear el siguiente comando:

```
  git add index.html
```

Una vez ejecutado el comando anterior podemos volver a lanzar el comando de **git status** y verificar ahora el nuevo estado del repositorio el cual debería de mostrar un mensaje como el siguiente:

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   index.html
#
```

En el log del estatus anterior nos dice que tenemos un cambio el cual falta por hacer COMMIT y que dicho cambio comprende un nuevo fichero llamado index.html.

Paso 4: Hacer COMMIT de cambios

Para hacer **COMMIT** de los cambios registrador o agregados al repositorio empleamos el siguiente comando:

```
git commit -m "Commit incial"
```

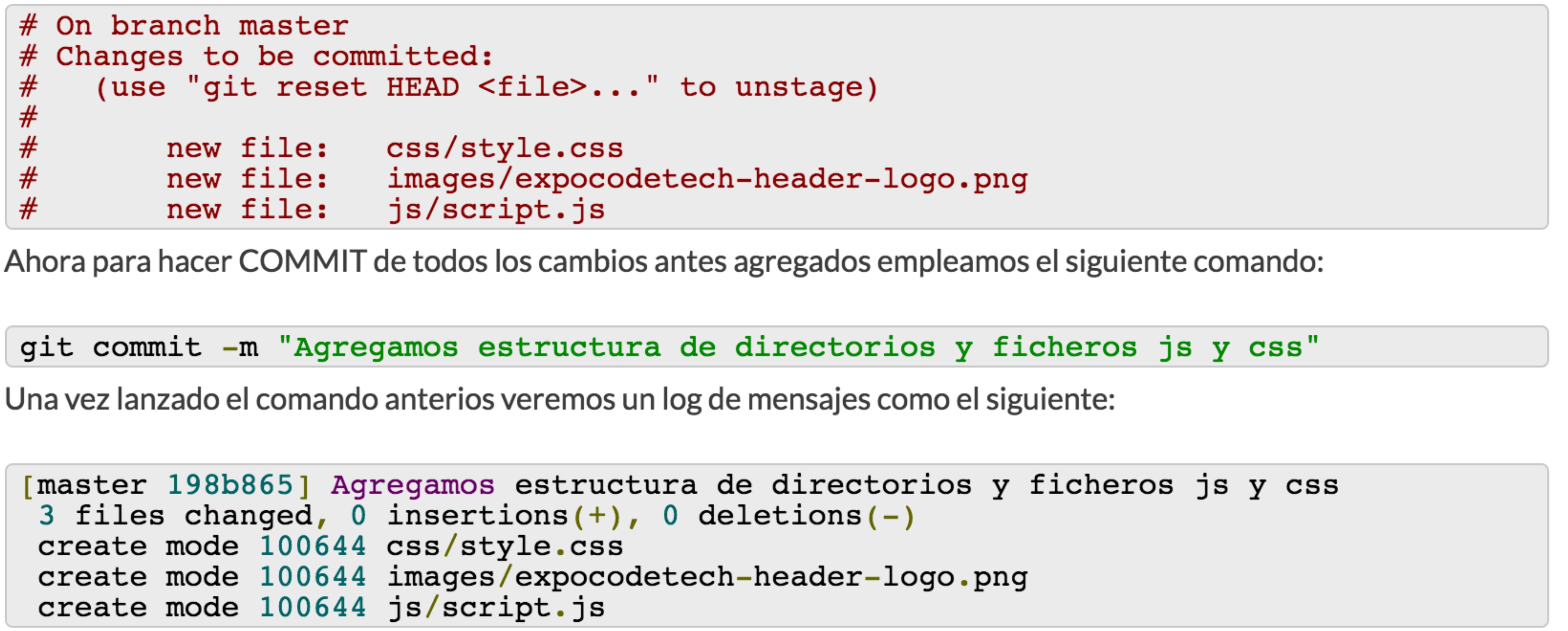
***Nota:** El parametro -m indica un mensaje el cual se indica entre comillas a la derecha de dicho parámetro, generalmente es usado para describir los cambios que se han hecho bajo dicho COMMIT*

Una vez lanzado el comando anterior veremos un mensaje de log como el siguiente, en el cual se especifican los cambios agregados al repositorio:

```
[master (root-commit) 2025d57] Commit inicial
1 file changed, 10 insertions(+)
create mode 100644 index.html
```

Ahora bien, generalmente las aplicaciones que desarrollamos mantiene mas de un fichero y seria demasiado tedioso agregar fichero a fichero al repositorio para poder llevar una gestión de versiones de los mismo, por lo tanto, para evitar agregar uno a uno empelamos el siguiente comando:

```
git add *
```



```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   css/style.css
#       new file:   images/expocodetech-header-logo.png
#       new file:   js/script.js
```

Ahora para hacer COMMIT de todos los cambios antes agregados empleamos el siguiente comando:

```
git commit -m "Agregamos estructura de directorios y ficheros js y css"
```

Una vez lanzado el comando anterios veremos un log de mensajes como el siguiente:

```
[master 198b865] Agregamos estructura de directorios y ficheros js y css
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 css/style.css
 create mode 100644 images/expocodetech-header-logo.png
 create mode 100644 js/script.js
```

Paso 5: Historial de cambios en GIT

Ahora que hemos realizados varios COMMIT de cambios al repositorio veamos como obtener el registro de todos los cambios realizados; para ello empleamos el siguiente comando:

```
git log
```

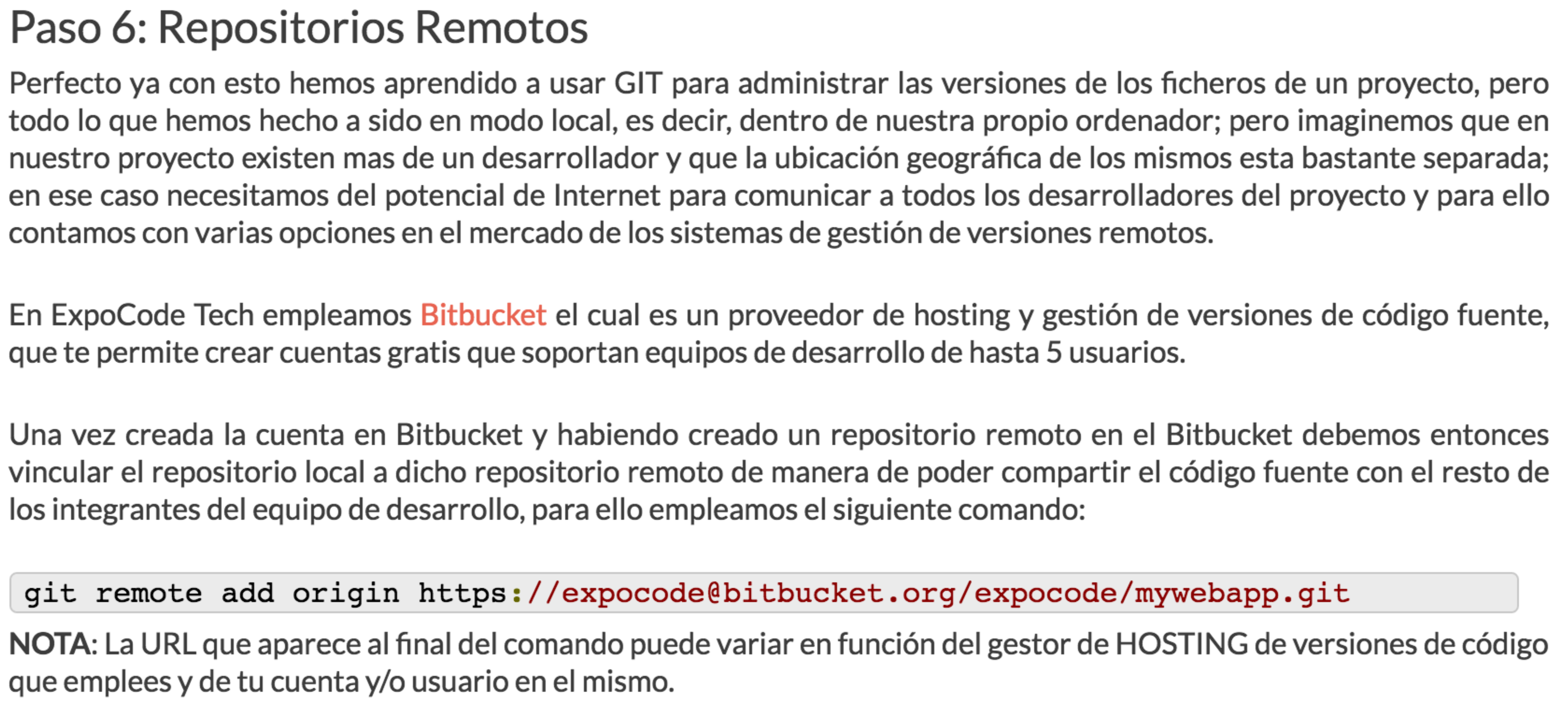
Al lanzar el comando anterior deberíamos de ver algo como lo siguiente:

```
commit 198b8657efebf93f07d8f2142157f4faed17bc65
Author: unknown <expocodetech@gmail.com>
Date:   Mon Feb 10 11:22:41 2014 +0100

    Agregamos estructura de directorios y ficheros js y css

commit 2025d577c9fae57790b026b39a7e562e17765cd4
Author: unknown <expocodetech@gmail.com>
Date:   Mon Feb 10 10:54:42 2014 +0100

    Commit inicial
```



Paso 6: Repositorios Remotos

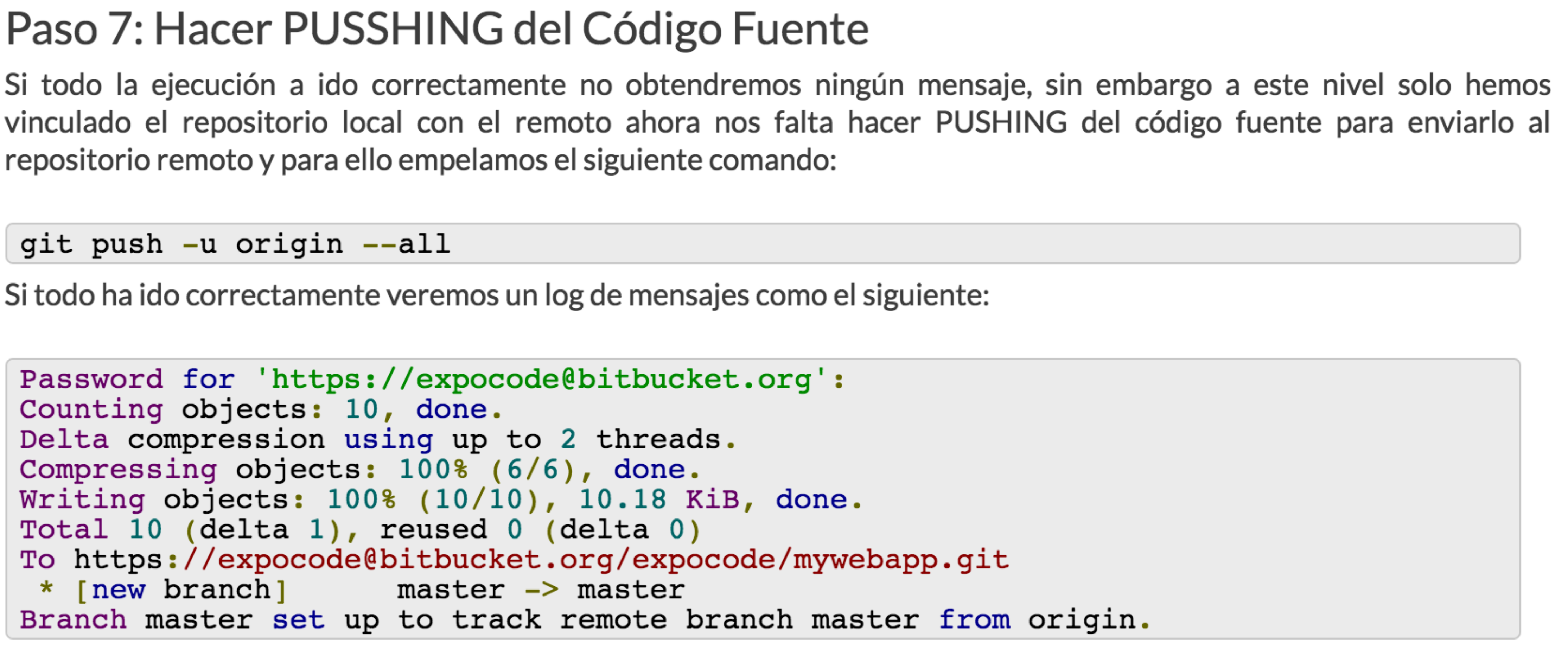
Perfecto ya con esto hemos aprendido a usar GIT para administrar las versiones de los ficheros de un proyecto, pero todo lo que hemos hecho a sido en modo local, es decir, dentro de nuestra propio ordenador; pero imaginemos que en nuestro proyecto existen mas de un desarrollador y que la ubicación geográfica de los mismos esta bastante separada; en ese caso necesitamos del potencial de Internet para comunicar a todos los desarrolladores del proyecto y para ello contamos con varias opciones en el mercado de los sistemas de gestión de versiones remotos.

En ExpoCode Tech empleamos Bitbucket el cual es un proveedor de hosting y gestión de versiones de código fuente, que te permite crear cuentas gratis que soportan equipos de desarrollo de hasta 5 usuarios.

Una vez creada la cuenta en Bitbucket y habiendo creado un repositorio remoto en el Bitbucket debemos entonces vincular el repositorio local a dicho repositorio remoto de manera de poder compartir el código fuente con el resto de los integrantes del equipo de desarrollo, para ello empleamos el siguiente comando:

```
git remote add origin https://expocode@bitbucket.org/expocode/mywebapp.git
```

NOTA: La URL que aparece al final del comando puede variar en función del gestor de HOSTING de versiones de código que emplees y de tu cuenta y/o usuario en el mismo.



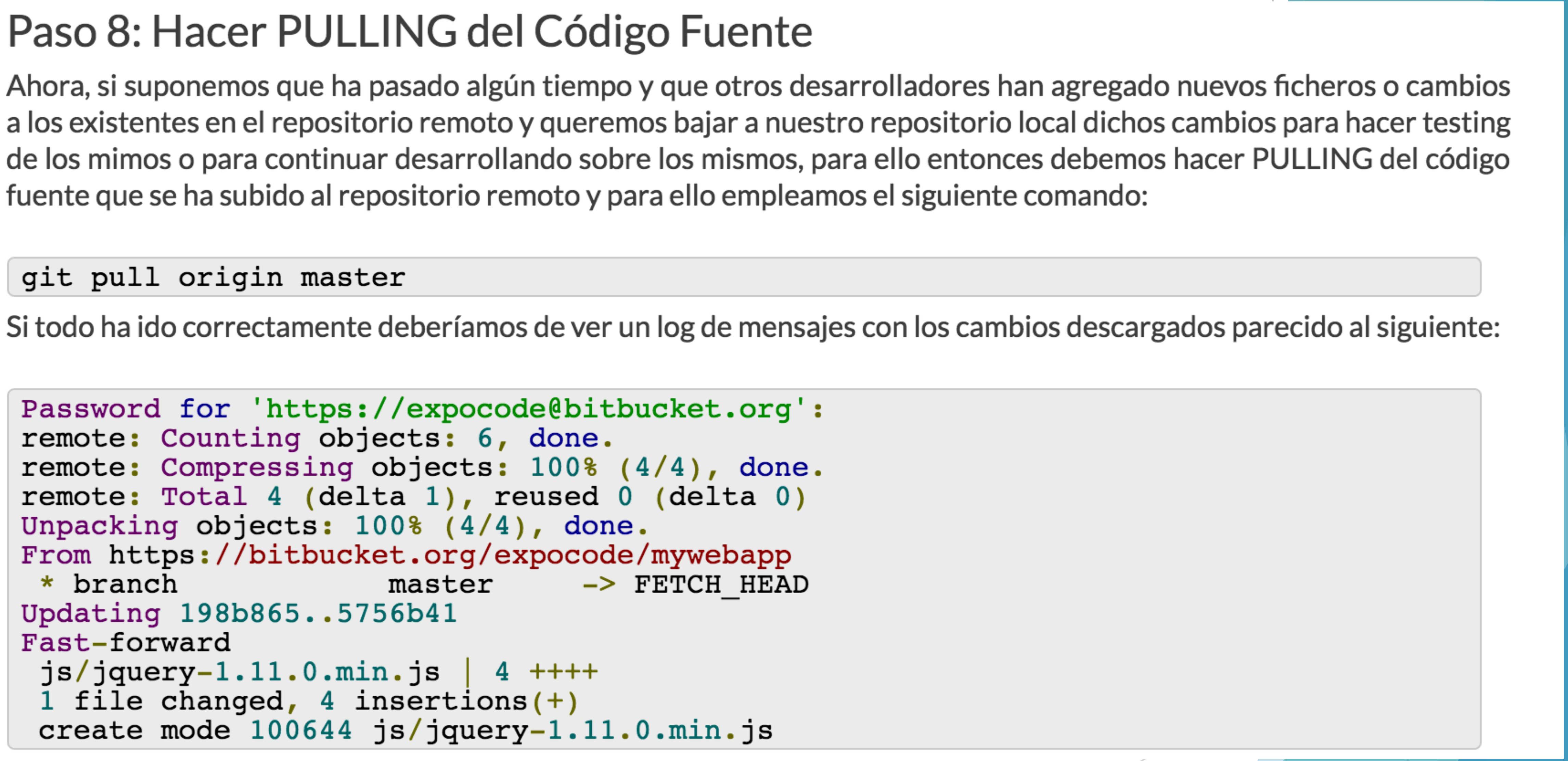
Paso 7: Hacer PUSSHING del Código Fuente

Si todo la ejecución a ido correctamente no obtendremos ningún mensaje, sin embargo a este nivel solo hemos vinculado el repositorio local con el remoto ahora nos falta hacer PUSHING del código fuente para enviarlo al repositorio remoto y para ello empelamos el siguiente comando:

```
git push -u origin --all
```

Si todo ha ido correctamente veremos un log de mensajes como el siguiente:

```
Password for 'https://expocode@bitbucket.org':
Counting objects: 10, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (10/10), 10.18 KiB, done.
Total 10 (delta 1), reused 0 (delta 0)
To https://expocode@bitbucket.org/expocode/mywebapp.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```



Paso 8: Hacer PULLING del Código Fuente

Ahora, si suponemos que ha pasado algún tiempo y que otros desarrolladores han agregado nuevos ficheros o cambios a los existentes en el repositorio remoto y queremos bajar a nuestro repositorio local dichos cambios para hacer testing de los mimos o para continuar desarrollando sobre los mismos, para ello entonces debemos hacer PULLING del código fuente que se ha subido al repositorio remoto y para ello empleamos el siguiente comando:

```
git pull origin master
```

Si todo ha ido correctamente deberíamos de ver un log de mensajes con los cambios descargados parecido al siguiente:

```
Password for 'https://expocode@bitbucket.org':
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From https://bitbucket.org/expocode/mywebapp
 * branch            master     -> FETCH_HEAD
Updating 198b865..5756b41
Fast-forward
 js/jquery-1.11.0.min.js | 4 ++++
 1 file changed, 4 insertions(+)
 create mode 100644 js/jquery-1.11.0.min.js
```