SOLUCIÓN EJ. GIT, GITHUB Y MARKDOWN

ADOLFO SANZ DE DIEGO

DICIEMBRE 2015

1 ACERCA DE

1.1 AUTOR

- **Adolfo Sanz De Diego**
  - Blog: asanzdiego.blogspot.com.es
  - Correo: asanzdiego@gmail.com
  - GitHub: github.com/asanzdiego
  - Twitter: twitter.com/asanzdiego
  - LinkedIn: in/asanzdiego
  - SlideShare: slideshare.net/asanzdiego

1.2 LICENCIA

- **Este obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0

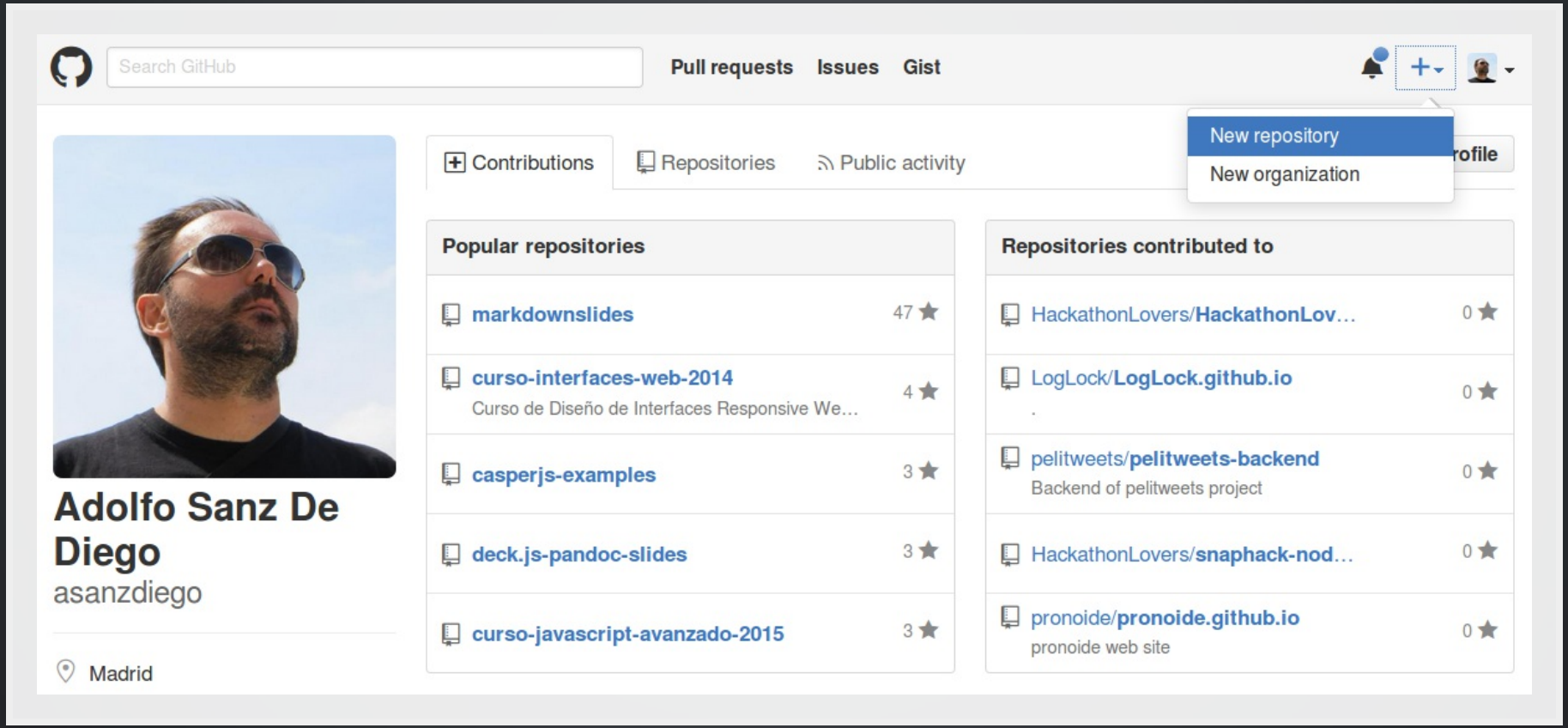
1.3 FUENTE

- Las slides y sus fuentes las podéis encontrar en:
  - https://github.com/asanzdiego/curso-git-github-markdown-2015

2 EJERCICIOS

2.1 REPOSITORIO CAMPUSCIFF (I)

1. Crear un repositorio en vuestro GitHub llamado **campusciff**.



Crear un repositorio

2.2 REPOSITORIO CAMPUSCIFF (II)

1. Clonar vuestro repositio en local.

```
git clone git@github.com:asanzdiego/campusciff.git
```

2.3 README

1. Crear (si no lo habéis creado ya) en vuestro repositorio local un documento **README.md**.

> *Notas: en este documento tendreís que ir poniendo los **comandos** que habéis tenido que utilizar durante todos los ejercicios y las **explicaciones y capturas de pantalla** que consideréis **necesarias**.*

2.4 COMMIT INICIAL

1. Añadir al README.md los comanddos utilizados hasta ahora y hacer un coomit inicial con el mensaje **commit inicial**.

```
git add .
git commit -m "commit inicial"
```

2.5 PUSH INICIAL

1. Subir los cambios al repositorio remoto.

```
git push origin master
```

2.6 IGNORAR ARCHIVOS (I)

1. Crear en el repositorio local un fichero llamado **privado.txt**.

```
touch privado.txt
```

1. Crear en el repositorio local una carpeta llamada **privada**.

```
mkdir privada
```

2.7 IGNORAR ARCHIVOS (II)

1. Realizar los cambios oportunos para que tanto el archivo como la carpeta sean ignorados por git.

```
echo "privado.txt" > .gitignore
echo "/privada" > .gitignore
git add .
git commit -m "añadido fichero .gitignore"
```

2.8 AÑADIR FICHERO 1.TXT

1. Añadir fichero **1.txt** al repositorio local.

```
touch 1.txt
git add .
git commit -m "añadido 1.txt"
```

2.9 CREAR EL TAG V0.1

1. Crear un tag **v0.1**.

```
git tag v0.1
```

2.10 SUBIR EL TAG V0.1

1. Subir los cambios al repositorio remoto.

```
git push --tag origin master
```

2.11 CREAR UNA RAMA V0.2

1. Crear una rama **v0.2**.

```
git branch v0.2
```

1. Posiciona tu carpeta de trabajo en esta rama.

```
git checkout v0.2
```

2.12 AÑADIR FICHERO 1.TXT

1. Añadir un fichero **2.txt** en la rama **v0.2**.

```
touch 2.txt
git add .
git commit -m "añadido 2.txt"
```

2.13 CREAR RAMA REMOTA V0.2

1. Subir los cambios al reposiorio remoto.

```
git push origin v0.2
```

2.14 MERGE DIRECTO

1. Posicionarse en la rama **master**.

```
git checkout master
```

1. Hacer un merge de la rama **v0.2** en la rama **master**.

```
git merge v0.2 -m "merge v0.2 sin conflictos"
```

2.15 MERGE CON CONFLICTO (I)

1. En la rama **master** poner **Hola** en el fichero **1.txt** y hacer commit.

```
git checkout master
echo "Hola" >> 1.txt
git add .
git commit -m "hola en 1.txt"
```

2.16 MERGE CON CONFLICTO (II)

1. Posicionarse en la rama **v0.2** y poner **Adios** en el fichero "1.txt" y hacer commit.

```
git checkout v0.2
echo "Adios" >> 1.txt
git add .
git commit -m "adios en 1.txt"
```

2.17 MERGE CON CONFLICTO (III)

1. Posicionarse de nuevo en la rama **master** y hacer un merge con la rama **v0.2**

```
git checkout master
git merge v0.2
vim 1.txt
git add .
git commit -m "arreglado merge en 1.txt"
```

2.18 LISTADO DE RAMAS

1. Listar las ramas con merge y las ramas sin merge.

```
git branch --merged
git branch --no-merged
```

2.19 ARREGLAR CONFLICTO

1. Arreglar el conflicto anterior y hacer un commit.

```
vim 1.txt
git add .
git commit -m "arreglado merge en 1.txt"
```

2.20 BORRAR RAMA

1. Crear un tag **v0.2**

```
git tag v0.2
```

1. Borrar la rama **v0.2**

```
git branch -d v0.2
```

2.21 LISTADO DE CAMBIOS

1. Listar los distintos commits con sus ramas y sus tags.

```
git config --global alias.list 'log --oneline --decorate --graph --all'
git list
```

2.22 CUENTA DE GITHUB

1. Poner una foto en vuestro perfil de GitHub.
2. Poner el doble factor de autentificación en vuestra cuenta de GitHub.
3. Añadir (si no lo habéis hecho ya) la clave pública que se corresponde a tu ordenador.

2.23 USO SOCIAL DE GITHUB

1. Preguntar los nombres de usuario de GitHub de tus compañeros de clase, búscalos, y sigueles.
2. Seguir los repositorios **campusciff** del resto de tus compañeros.
3. Añadir una estrella a los repositorios **campusciff** del resto de tus compañeros.

2.24 CREAR UNA TABLA

1. Crear una tabla de este estilo en el fichero **README.md** con la información de varios de tus compañeros de clase:

NOMBRE	GITHUB
Nombre del compañero 1	enlace a github 1
Nombre del compañero 2	enlace a github 1
Nombre del compañero 3	enlace a github 3

2.25 COLABORADORES

1. Poner a github.com/asanzdiego como colaborador del repositorio **campusciff**

2.26 CREAR UNA ORGANIZACIÓN

1. Crear una organización llamada **campusciff-tunombredeusuariodegithub**

2.27 CREAR EQUIPOS

1. Crear 2 equipos en la organización **campusciff-tunombredeusuariodegithub**, uno llamado **administradores** con más permisos y otro **colaboradores** con menos permisos.
2. Meter a github.com/asanzdiego y a 2 de vuestros compañeros de clase en el equipo **administradores**.
3. Meter a github.com/asanzdiego y a otros 2 de vuestros compañeros de clase en el equipo **colaboradores**.

2.28 CREAR UN INDEX.HTML

1. Crear un index.html que se pueda ver como página web en la organización.

2.29 CREAR PULL-REQUESTS

1. Hacer 2 forks de 2 repositorios **campusciff-tunombredeusuariodegithub.github.io** de 2 organizaciones de las que no seais ni administradiores ni colaboradores.
2. Crearos una rama en cada fork.
3. En cada rama modificar el fichero **index.html** añadiendo vuestro nombre.
4. Con cada rama hacer un pull-request.

2.30 GESTIONAR PULL-REQUESTS

1. Aceptar los pull-request que lleguen a los repositorios de tu organización.