Informe de la práctica de Laboratorio Nº2: Control de versiones con Git. Implantación en Heroku.

Autor: Adán Rafael López Lecuona.

1- Ruby 3 Tutorial: Version control with Git

Para el control de versiones Git primero me creé una cuenta en Github y introduje un nombre de usuario, una dirección de correo y una contraseña. Luego creé un nuevo repositorio en la página llamado DSI con el nombre de nuestro proyecto.

-Ahora configuramos nuestro nombre e email en nuestra máquina virtual (esto se hace si no te has registrado antes en Git en tu máquina) mediante los comandos,:

```
$ git config --global user.name "alu3954"
```

```
$ git config --global user.email x-ando@hotmail.com
```

Configuramos el alias checkout y el editor:

```
$ git config --global alias.co checkout
```

```
$ git config --global core.editor "subl -w"
```

-El editor me da problemas ya que los cambié a "mate -w" y lo volví a poner a "subl -w" y me seguía dando problemas.

-Inicializamos un nuevo repositorio mediante el comando dentro de nuestro proyecto:

```
$ git init
```

-Se producen una serie de archivos y entre ellos está el .gitignore que cambio editándolo con vi .gitignore introduciendo, en el cuál se introducen los archivos que queremos ignorar para git:

```
# Ignore other unneeded files.
```

```
doc/
```

```
*.swp
```

```
*~
```

.project

.DS_Store

.idea

-Para añadir todos los ficheros utilicé el comando add, podemos añadir específicamente también.

$ git add .

-Para ver el los archivos modificados utilizamos:

$ git status

```
On branch master

#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README.rdoc
#       new file:   Rakefile
.
```

-Luego si queremos guardar nuestros cambios lo hacemos mediante el comando:

$ git commit -m "Initial commit"

Con el commit observamos nuestras inserciones, archivos modificados y eliminados.

-Si queremos listar la lista de commits(id commit, autor, email, y fecha)hacemos:

$ git log

-Después borramos el app/controllers.

$ ls app/controllers/

$ rm -rf app/controllers/

$ ls app/controllers/

$ git checkout -f --> nos aparece el directorio perdido

GITHUB

Para poner un proyecto en el control de versiones Git utilizamos Github para poder subir nuestro código. Lo primero es crearnos una cuenta en Github.

Crear un nueva contraseña SSH keys para un determinado repositorio creado en Github.

Para ello tengo que generar una contraseña y copiarla en la sección de Github de SSH Keys y añadiendo una nueva y copiando ahí la clave generada antes.

Me da problemas al poder añadir la contraseña.

Para añadir a un repositorio remoto utilizo el comando:

$ git remote add origin git@github.com:<alu3954>/first_app.git

$ git push -u origin master ---> para añadir los cambios a la rama master del proyecto en Github.

$ git remote add origin git@github.com:railstutorial/first_app.git

Branch

Seguimos una serie de pasos para el branch.

$ git checkout -b modify-README

$ git branch

$ git mv README.rdoc README.md

$ subl README.md

Nos aparecen los archivos que hemos modificado o renombrado.

$ git status.

Nos aparecen los cambios realizados, las inserciones y las eliminaciones.

$ git commit -a -m "Improve the README file"

Merge

$ git checkout master

$ git merge modify-README

$ git branch -d modify-README

Para enviar los cambios a git realizamos el comando:

$git push.

Me da problemas al poder hacer push ya que me sale clave invalidada.

2- Control de versiones para proyectos Django

Django es un entorno de alto nivel basado en Python. Con Django instalado se crea una estructura así por defecto:

ejemplo/

_init.py

Manage.py

Setting.py

Urls.py

Se crea una aplicación con esta estructura de directorios.

Para crear un proyecto de Django se realiza de forma similar a un proyecto en Ruby on Rails mediante el control de versiones git, para poder subirlo a Github.

3- Implantación de Heroku

Para implantar Heroku me creé una cuenta en la página oficial de Heroku http://www.heroku.com/ y lo instalé en la máquina virtual.

Heroku (Ruby on Rails).

En la carpeta que creé llamada proyectos_RAILS creo una nueva aplicación heroku mediante el comando:

$heroku create.

```
Creating simple-spring-9999... done, stack is cedar
http://simple-spring-9999.herokuapp.com/ | git@heroku.com:simple-spring-9999.git
Git remote heroku added
```

Modificamos el archivo Gemfile para el grupo producción añadiendo al archivo esto:

```
source 'https://rubygems.org'

gem 'rails', '3.2.12'

group :development do
  gem 'sqlite3', '1.3.5'
end


# Gems used only for assets and not required
# in production environments by default.
group :assets do
  gem 'sass-rails',   '3.2.5'
  gem 'coffee-rails', '3.2.2'

  gem 'uglifier', '1.2.3'
end

gem 'jquery-rails', '2.0.2'

group :production do
  gem 'pg', '0.12.2'
end
```

Para migrar nuestra base de datos lo hacemos mediante:

```
$ heroku run rake db:migrate
```

Tenemos que sincronizar nuestra base de datos mediante syncdb.

Para iniciar una cuenta con heroku debemos poner el comando:

$heroku login en el cuál nos pedirá el correo y la contraseña.

Para subir a Heroku mediante Git realizamos el comando:

$git push heroku master

Para abrir nuestro proyecto con heroku en nuestro navegador se realiza mediante el comando:

$heroku open

Podemos ver los registros de heroku mediante el comando:

$heroku logs

Podemos usar tanto Django y Ruby on Rails en heroku mediante el control de versiones de Git .

Con heroku a podemos subir nuestra app a la red para poder visualizarla en la nube, tanto con Django como con Ruby on Rails.