Ejercicio 5.1 Sistema de control de versiones con **git**

Es posible que su sistema ya tenga instalado **git**. El comando `which git` le mostrará si está presente en el sistema. Si no lo está, usted puede obtener las fuentes, compilarlas e instalarlas, pero suele ser mucho más fácil instalar los paquetes binarios precompilados; su instructor puede ayudarlo a identificar los paquetes necesarios en caso que no estén instalados, o que no puedan ser instalados con alguno de los comandos siguientes:

```
$ sudo yum install git*
$ sudo zypper install git*
$ sudo apt-get install git*
```

de acuerdo a su distribución en particilar.

Comencemos a ver cómo **funciona git** y cuán fácil es usarlo. Por el momento nos limitaremos a crear nuestro proyecto local propio.

1. Primero crearemos un directorio de trabajo y luego inicializaremos **git** para que trabaje con él:

   ```
   $ mkdir git-test
   $ cd git-test
   $ git init
   ```

2. La inicialización del proyecto crear un directorio `.git`, el cual contenerá toda la información del control de versiones; los directorios principales que se incluyen en el proyecto permanecerán intactos. El contenido inicial del directorio luce así:

   ```
   $ ls -l .git
   total 40
   drwxrwxr-x 7 coop coop 4096 Dec 30 13:59 ./
   drwxrwxr-x 3 coop coop 4096 Dec 30 13:59 ../
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 branches/
   -rw-rw-r-- 1 coop coop   92 Dec 30 13:59 config
   -rw-rw-r-- 1 coop coop   58 Dec 30 13:59 description
   -rw-rw-r-- 1 coop coop   23 Dec 30 13:59 HEAD
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 hooks/
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 info/
   drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 objects/
   drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 refs/
   ```

   Más adelante describiremos el contenido de este directorio y sus subdirectorios, los que en su mayor parte comienzan vacíos.

3. A continuación crearemos un archivo y lo agregaremos al proyecto:

   ```
   $ echo some junk > somejunkfile
   $ git add somejunkfile
   ```

4. Podemos ver el estado actual de nuestro proyecto con:

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

   Note que la salida anterior está mostrando que el archivo está **preparado** pero todavía no ha sido **registrado**.

5. Ahora modifiquemos el archivo y luego veamos la historia de las diferencias:


```
$ echo another line >> somejunkfile
$ git diff
diff --git a/somejunkfile b/somejunkfile
index 9638122..6023331 100644
--- a/somejunkfile
+++ b/somejunkfile
@@ -1 +1,2 @@
 some junk
+another line
```

6. Para registrar los cambios en el repositorio hacemos lo siguiente:

```
$  git commit -m "My initial commit" --author="A Genius <a_genius@linux.com>"
Created initial commit eafad66: My initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 somejunkfile
```

   La opción `--author` es opcional. Si usted no especifica un mensaje de identificación con la opción `-m` al realizar el commit, usted será llevado a un editor para que ingrese el contenido correspondiente. Usted **debe** hacer esto o el commit será rechazado. Se elejirá el editor que esté configurado en la variable de ambiente `EDITOR` , la cual puede ser reemplazada con la configuración de `GIT_EDITOR`.

7. Puede ser tedioso agregar la información del autor cada vez que haga un commit. Es posible hacerlo de forma automática con:

```
$ git config user.name "Another Genius"
$ git config user.email "b_genius@linux.com"
```

   lo cual se utilizará en el próximo commit.

8. Es posible ver la historia con:

```
$ git log
commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc
Author: A Genius <a_genius@linux.com>
Date:   Wed Dec 30 11:07:19 2009 -0600

    My initial commit
```

   y puede ver la información que hay ahí. Notará que el string largo haxadecimal es un identificador único de 160 bit y 40 dígitos, el cual corresponde al **número de commit**. **Git** trabaja con estos identificadores y no con los nombres de archivos.

9. Ahora usted es libre de modificar el archivo existente y agregar archivos nuevos con `git add`. Pero ellos estarán en estado staged (modificados y preparados para ser enviados al repositorio) hasta que usted haga otro `git commit`.

10. Hasta aquí vamos bien, aunque hemos visto solo la superficie de git.

