02_gestion_de_repositorios

October 15, 2020

1 Gestión de repositorios.

1.1 Estructura de un repositorio de *Git*.

Un repositorio de *Git* es un directorio el cual a su vez contiene un subdirectorio `.git` con una estructura similar a la siguiente:

```
.git
   config
   description
   FETCH_HEAD
   HEAD
   hooks
   index
   info
   logs
       HEAD
       refs
   objects
       info
       pack
   ORIG_HEAD
   refs
        heads
        remotes
        tags
```

1.2 Incialización de un repositorio.

El comando `git init` permite crear la estructura básica de un subdirectorio `.git` en un directorio dado mediante la siguiente sintaxis:

```
git init <ruta>
```

Donde:

- `<ruta>` es la ruta del directorio a inicializar. Si no se indica la ruta, el comando creará el directorio `.git` en el directorio desde el cual se ejecuta el comando.

Para mayor información sobre `git init` es posible consultar la siguiente liga:

https://git-scm.com/docs/git-init

Ejemplo:

- La siguiente celda creará el directorio `demo`.

```
[ ]: mkdir demo
```

```
[ ]: tree demo
```

- La siguiente celda inicializará al directorio `demo`como un repositorio de *Git*.

```
[ ]: git init demo
```

- La siguiente celda ejecutará el comando `tree` para el subdirectorio `demo/.git`, mostrando su estructura.

```
[ ]: tree demo/.git
```

1.3 Estado de un repositorio.

El estado de un respositorio corresponde al estado de todos los archivos a los que *Git* está dando seguimiento dentro de dicho repositorio.

La característica principal de un gestor de versiones es la de poder conservar un registro detallado del estado de un repositorio en un momento específico.

1.3.1 Los "commits".

Guardar el estado de un repositorio de forma continua como si se tratara de una película resulta ser poco eficiente y costoso en recursos de almacenamiento. Es por ello que los gestores de verisones como *Git* han optado por conservar el estado de un repositorio sólo cuando el usuario decide asentarlos. A dicha accion se le conoce como hacer un "commit".

Los "commits" son la base del control de versiones.

1.3.2 Despliegue del estado de un repositorio.

El comando `git status` permite conocer el estado de un repositorio mediante la siguiente sintaxis:

`git status <ruta>`

Donde:

- `<ruta>` es la ruta del directorio a consultar. Si no se indica la ruta, el comando consultará el estado del directorio desde el cual se ejecuta el comando.

En caso de que la ruta no corresponda a un directorio que contenga una estructura de directorio `.git` adecuada, se generará unmensaje de error.

Para mayor información sobre `git status` es posible consultar la siguiente liga:

https://git-scm.com/docs/git-status

Ejemplos:

- La siguiente celda mostrará el estado del directorio `demo`.

[ ]:
```
git status demo
```

- La siguiente celda mostrará el estado del directorio en el que se encuentra esta notebook.

[ ]:
```
git status
```

1.4 Rastreo de los archivos dentro de un repositorio.

Git tiene la capacidad de dar seguimiento a los cambios de cada archivo dentro del repositorio, identificando a cada uno de ellos como un objeto.

Para que *Git* pueda dar seguimiento a un objeto, es necesario darlo de alta en su índice. Una vez que un archivo es dado de alta, *Git* puede evaluar si dicho archivo ha sido modificado.

1.4.1 Área de preparación (staging area).

Git permite definir puntualmente aquellos archivos cuyos cambios serán asentados en un commit.

El área de preparación (staging area) permite al usuario registrar aquellos cambios que serán incluidos en el commit.

1.4.2 Estado de un archivo.

- **Sin seguimiento** (untracked), lo que implica que el archivo no está en el índice del repositorio y por lo tanto sus cambios no serán rastreados por *Git*.
- **En seguimiento** (tracked) lo que implica que el archivo está en el índice del repositorio.
  - **Modificado** (modified) lo que implica que un archivo ha sido modificado, eliminado o ha cambiado de nombre.
  - **No modificado** (unmodified).

El comando `git status` da un resumen de: * Aquellos archivos creados después del último commit y que no tienen seguimiento. * Aquellos archivos que están en seguimiento y que fueron modificados después del último commit.

1.4.3 Seguimiento y rastreo de archivos.

El comando`git add` es el encargado tanto de dar seguimiento a un archivo como de rastrear y registrar los cambios en el área de preparación. Su sintaxis es la siguiente:

`git add <patrón o ruta> <opciones>`

Donde:

- `<patrón o ruta>` puede ser la ruta a un archivo o directorio específico o un patrón que identifique a más de un archivo.
- `<opciones>` este comando cuenta con varias opciones útiles.

En caso de que el archivo no haya estado en seguimiento, este y su contenido serán indexados. En caso de que el archivo ya se encuentre en seguimiento, se actualizará el registro de sus modificaciones.

https://git-scm.com/docs/git-add

La opción `--all`. Esta opción aplicará la ejecuciónde `git add` a todos los archivos del repositorio.

Ejemplo:

- La siguiente celda moverá el shell al directorio `demo`.

[ ]:
```
cd demo
```

- La siguiente celda creará a los archivos `archivo-1`, `archivo-2` y `archivo-3`.

[ ]:
```
touch archivo-1 archivo-2 archivo-3
```

[ ]:
```
ls -a
```

- En este momento los archivos recién creados no están en seguimiento.

[ ]:
```
git status
```

- La siguiente celda indexará a `archivo-1` y lo registrará en el área de preparación.

[ ]:
```
git add archivo-1
```

[ ]:
```
git status
```

- La siguiente celda indexará al resto de los archivos en el repositorio y los registrará en el área de preparación.

[ ]:
```
git add --all
```

[ ]:
```
git status
```

1.5 Listado de objetos en el índice del repositorio.

Para saber cuáles son los objetoss indexados en un repositorio se utiliza el comando `git ls-files` con la siguiente sintaxis:

```
git ls-files
```

La documentación de referencia de `git ls-files` está disponible en:

https://git-scm.com/docs/git-ls-files

Ejemplo:

La siguiente celda desplegará el listado de archivos indexados del repositorio.

[ ]:
```
git ls-files
```

[ ]:
```
git ls-files -s
```

1.6 El archivo `.gitignore`.

Es común que un desarrollador quiera evitar indexar en el repositorio archivos tales como:

- Archivos temporales.
- Credenciales y contraseñas.
- Bibliotecas que resulten redundantes.
- Datos de prueba.

Git permite definir una "lista negra" de archivos que serán ignorados por los comandos de `git` de forma automática. Esta lista negra se define mediante un archivo de texto llamado `.gitignore`, el cual va en el directorio principal del repositorio.

Este archivo contiene un listado en forma de columma ya sea de los nombres de los archivos o del patrón correspondiente a ciertos archivos que deberán ser ignorados.

La documentación de referencia de `.gitignore` está disponible en:

https://git-scm.com/docs/gitignore

Ejemplo:

- La siguiente celda creará al archivo `.gitignore` listando al nombre `invisible`.

[ ]:
```
echo "invisible" > .gitignore
```

[ ]:
```
cat .gitignore
```

- La siguiente celda creará al archivo `invisible`.

[ ]:
```
touch invisible
```

- El sistema de archivos del repositorio tiene los archivos.
  - `archivo-1`
  - `archivo-2`
  - `archivo-3`
  - `.gitignore`
  - `invisible`

[ ]:
```
ls -a
```

- El comando `git status` identificará al archivo `.gitignore`, pero no a `invisible`.

[ ]:
```
git status
```

- Lo mismo ocurre con `git add`.

[ ]:
```
git add --all
```

[ ]:
```
git status
```

1.7 Confirmando los cambios en un repositorio ("commit").

Una vez que el usuario realiza todas las operaciones necesarias en el área de preparación (staging area) está listo para hacer un commit.

Nota: Para poder realizar un commit es necesario que el usuario defina al menos los campos `user.name` y `user.email` en la configuración del ámbito `--global`.

1.7.1 Flujo de un commit.

- Cada objeto en el área de preparación es evaluado.
  - En caso de que sea un nuevo objeto, este será incluído en el estado del nuevo commit.
  - En caso de que el objeto ya esté registrado en un commit previo y haya sido modificado, se enumerarán y registrarán las modificaciones con respecto a la versión previa.
  - Todos los cambios serán registrados.
  - Se registrarán los cambios en el estado del repostorio, así como el momento exacto del commit.
  - Al commit se le asignará un identificador único que corresponde a un número hexadecimal.
  - Se vaciará el área de preparación.
  - Se desplegará un resumen del commit.

1.7.2 El comando `git commit`.

El comando `git commit` es el encargado de confirmar los cambios en el repositorio. Cada commit requiere que se haga un comentario descriptivo de las modificaciones.

La sintaxis es:

```
git commit <opciones> <archivos>
```

Donde:

- `<archivos>` permite definir un listado de archivos que serán tomados en commit de forma similar a `git add`.
- `<opciones>` las cuales definen diversas acciones para el commit.

En caso de que se ejecute el comando `git commit` sin definir archivos u opciones, se abrirá un editor de textos (*vim*) en el cual se deberá de escribir un mensaje descriptivo del commit y una vez guardado, se realizará el commit tomando en cuanta todos los objetos definidos en el área de preparación.

La documentación de referencia de `git commit` está disponible en:

https://git-scm.com/docs/git-commit

Nota: En caso de que no existan cambios en los objetos del área de preparacion, no se hará el commit.

La opción `--all` o `-a`. Esta opción busca y registra cambios en todos los objetos indexados del repositorio y los guarda en el área de preparación antes de hacer el commit. La opción `-a` no tomará en cuenta a archivos recién creados y que no hayan sido indexados.

La sintaxis puede ser:

```
git commit -a
```

o

```
git commit --all
```

La opción `--message` o `-m` . Esta opción permite añadir un mensaje descriptivo y evita que se abra el editor de texto.

La sintaxis puede ser:

```
git commit -m <mensaje>
```

o

```
git commit --message=<mensaje>
```

Donde:

- `<mensaje>` es una cadena de caracteres que en caso de incliur espacios,d ebe de estar encerrada entre comillas.

Ejemplo:

La siguiente celda realizará un commit del repositorio actual añadiendo el mensaje descriptivo `"primer commit`.

[ ]:
```
git commit -m "primer commit"
```

[ ]:
```
git status
```

1.7.3 Estructura del mensaje del commit.

Una vez hecho el commit, *Git* regesa un mensaje como el siguiente:

‘`bash [master (root-commit) 3c76e61] primer commit  4 files changed, 1 insertion(+)  create mode 100644 .gitignore  create mode 100644 archivo-1  create mode 100644 archivo-2  create mode 100644 archivo-3`

Este mensaje se compone de: * Un encabezado que describe al commit. * Un listado de las adiciones o modificaciones de los objetos en el espacio de preparación.

El encabezado de un mensaje de commit. El encabezado del mensaje de commit tiene una estructura como la siguiente:

`[<rama> (<tipo de commit>) <identificador corto>] <mensaje>`

Donde:

- `<rama>` es la rama del repositorio en la que se realzia el commit. La rama que se crra pord efecto en un repositorio es `master`.
- `<tipo de commit>` este dato es opcional y cuando se realiza el primer commti de un repoitorio despliega el mensaje “‘root-commit“‘.
- `<identificador corto>`, el cual es un número hexadecimal de 7 dígitos que corresponde a los primeros 7 dígitos del número identificador completo del commit.

1.8 Eliminación de un objeto.

El comando `git rm` se utiliza para eliminar a un objeto tanto del repositorio como de su índice.

```
git rm <archivo> <opciones>
```

Ejemplo:

- El archivo `archivo-3` existe tanto en el respositorio como en el índice.

[ ]:
```
ls -a
```

[ ]:
```
git ls-files
```

- A continuación se utilizará el comando `git rm` para eliminar al archivo tanto del repositorio como de su índice.

[ ]:
```
git rm archivo-3
```

[ ]:
```
ls -a
```

[ ]:
```
git ls-files
```

[ ]:
```
git status
```

- Se hará un nuevo commit con la descripción `segundo commit`.

[ ]:
```
git commit -m "segundo commit"
```

1.8.1 La opciones `--cached` y `-f`.

Es posible que existan archivos registrados en el área de preparación y que deban de ser eliminados antes de realizar un commit.

La opción `--cached` del comando `git rm` eliminará al objeto del índice, pero no al archivo en el repositorio.

La opción `-f` del comando `git rm` eliminará al objeto del índice y al archivo en el repositorio.

Ejemplo:

- Se creará al archivo `archivo-4` y se registrará en el área de preparación.

[ ]:
```
touch archivo-4
```

[ ]:
```
git add archivo-4
```

[ ]:
```
git status
```

- Debido a que no se ha hecho un `commit` para `archivo-4`, el comando `git rm archivo-4` no funcionará.

[ ]:
```
git rm archivo-4
```

- Utilizando la opción `--cached`, se eliminará al objeto del índice, pero se preservará en el directorio.

[ ]:
```
git rm archivo-4 --cached
```

[ ]:
```
ls -a
```

[ ]:
```
git status
```

- Se hará un nuevo commit con la descripción `tercer commit`. En este caso, como no hay cambios, el commit no se realizará.

[ ]:
```
git commit -m "tercer commit"
```

Se enviará nuevamente al archivo `archivo-4` al área de preparación.

[ ]:
```
git add archivo-4
```

[ ]:
```
git status
```

- Ahora se eliminará al archivo y al objeto `archivo-4`.

[ ]:
```
git rm archivo-4 -f
```

[ ]:
```
ls -a
```

[ ]:
```
git ls-files
```

1.9 Despliegue de la historia de un repositorio o un archivo.

El comando `git log` permite desplegar la historia de un repositorio o de un archivo.

La documentación de referencia de `git log` está disponible en:

https://git-scm.com/docs/git-log

1.9.1 Despliegue de la historia de un repositorio.

La sintaxis para desplegar la historia de un repositorio es la siguiente:

```
git log
```

- Para un archivo se utiliza la siguiente sintaxis:

```
git log <archivo>
```

El resutado es un listado de todos los commits empezando por el más reciente con los siguientes datos.

- El número completo del identficador del commit y la rama del commit.
- El nombre del autor según el campo `user.name` de la configuración.
- La dirección de correo del autor según el campo `user.emal` de la configuración.

Ejemplo:

- La siguiente celda mostrará la historia de los commits realizados en el repositorio `demo`.

[ ]:
```
git log
```

1.9.2 Despliegue de la historia de un archivo.

La sintaxis para desplegar la historia de un archivo es:

`git log <archivo>`

El resutado es un listado de los commits en los que el archivo en cuestión fue modificado.

Ejemplo:

- La siguiente celda mostrará la historia de los commits en los que se modificó el archivo `.gitignore` en el repositorio `demo`.

[ ]:
```
git log .gitignore
```

1.9.3 La opción `--oneline`.

Esta opción permite desplegar un listado resumido en una sola línea con la siguiente estructura:

`<identificador corto> (HEAD -> <rama>) <mensaje>`

Donde:

- `<rama>` es la rama del repositorio en la que se realzia el commit. La rama que se crra pord efecto en un repositorio es `master`.
- `<tipo de commit>` este dato es opcional y cuando se realiza el primer commti de un repoitorio despliega el mensaje `root-commit`.
- `<identificador corto>`, el cual es un número hexadecimal de 7 dígitos que corresponde a los primeros 7 dígitos del número identificador completo del commit

Ejemplos:

- La siguiente celda mostrará la historia de los commits realizados en el repositorio `demo` con una descripción corta.

[ ]:
```
git log --oneline
```

- La siguiente celda mostrará la historia de los commits en los que el archivo `archivo-1` del repositorio `demo` fue modificdo con una descripción corta.

[ ]:
```
git log --oneline archivo-1
```

1.10 El comando `git show`.

El comando `git show` permite mostrar la información del commit más reciente, incluyendo un resumen de las modificaciones a los archivos registradas en dicho commit.

`git show <ruta>`

Donde: * `ruta` es la ruta a un archivo. En caso de que no se indique la ruta, se traerá la información de todos los archivos modificados en el commit.

La documentación de referencia de `git show` está disponible en:

https://git-scm.com/docs/git-show

Ejemplo:

- La siguiente celda mostrará los detalles registrados en el commit más reciente.

[ ]:
```
git show
```

- La siguiente celda mostrará los detalles registrados en el commit más recientes afectando al archivo `archivo-1`. En vista de que no hubo afectaciones, no regresará nada.

[ ]:
```
git show archivo-1
```

1.11 Comparación entre archivos de distintos commits.

El comando `git diff`permite entre otras cosas visualizar las diferencias de uno o varios archivos entre un commit y otro.

La sintaxis de este comando es:

```
git diff <identificador 1> <identificador 2>
```

Donde:

- `<identificador 1>` e `<identificador 2>` son los dígitos iniciales de los números identificadores de algún commit realizado. El número de dígitos es arbitrario, siempre que el commit pueda ser identificado plenamente.

El resultado es un listado de cada objetos que presente variaciones entre un commit y otro, así como la descripción de dichas diferencias.

La documentación de referencia de `git diff` está disponible en:

https://git-scm.com/docs/git-diff

Ejemplo:

- la siguiente celda añadirá una línea de texto al archivo `archivo-1`.

[ ]:
```
echo Hola > archivo-1
```

[ ]:
```
cat archivo-1
```

- La siguiente celda traerá al área de preparación aquellos objetos que hayan sido modificados posteriormente al último commit y hará un nuevo commit con el mensaje `cuarto commit`

[ ]:
```
git commit -am "cuarto commit"
```

- La siguiente celda enlistará de forma corta todos los commits. El resultado será similar a los siguiente:

```
a1c53ed (HEAD -> master) cuarto commit
52c9f8c segundo commit
e71f6f8 primer commit
```

Nota: Cabe aclarar que los números identificadores siempre serán distintos en cada caso.

[ ]:
```
git log --oneline
```

- Para obtener un listado de diferencias entre el commit con descripción `cuarto commit` (en este ejemplo correspondería al identificador `ba397e8`) y el que tiene la descripción `primer commit` (en este ejemplo correspondería al identificador `ed7d117`).

La operación sería similar a lo siguiente.

```
> git diff ba397e8 ed7d117
```

Y el resultado sería similar al siguiente.

```
diff --git a/archivo-1 b/archivo-1
index a19abfe..e69de29 100644
--- a/archivo-1
+++ b/archivo-1
@@ -1 +0,0 @@
-Hola
diff --git a/archivo-3 b/archivo-3
new file mode 100644
index 0000000..e69de29
```

Nota: Cabe hacer notar que cada objeto del índice tiene su propio número identificador.

[ ]:
```
git diff 0e768b7 6e1e0fa # NOTA: Es necesario sustituir los números␣
↪identificadores.
```

[ ]:
```
git diff 6e1e0fa 0e768b7
```

1.11.1 Visualización de cambios de un archivo específico en distintos commits.

Para conocer las diferencias

La sintaxis de este comando es:

```
git diff <identificador 1> <identificador 2> <ruta>
```

Donde:

- `<identificador 1>` e `<identificador 2>` son los dígitos iniciales de los números identificadores de algún commit realizado. El número de dígitos es arbitrario, siempre que el commit pueda ser identificado plenamente.
- `<ruta>` es la ruta del archivo específico dentro del respositorio.

Ejemplo:

- Para conocer los cambios ocurridos en el archivo `archivo-1` entre el commit `ed7d11` y el commit `ba397e8` se ejecutaría lo siguiente:

```
> git diff ed7d117 ba397e8  archivo-1
```

- Y el resultado sería similar al siguiente:

```
diff --git a/archivo-1 b/archivo-1
index e69de29..a19abfe 100644
--- a/archivo-1
+++ b/archivo-1
@@ -0,0 +1 @@
+Hola
```

[ ]:
```
git diff 6e1e0fa 0e768b7 archivo-1 # NOTA: Es necesario sustituir los números␣
↪identificadores.
```