03_gestion_de_ramas

October 15, 2020

1 Gestión de ramas.

Otra de las características primordiales de los sistemas de gestión de versiones es su capacidad de poder separar flujos de trabajo en "ramas", la cuales pueden bifurcarse de una rama principal e incluso podrían converger en otro momento.

La combinación de commit y ramas permiten crear flujos de trabajo colaborativos muy eficientes en el desarrolo de un proyecto de software.

Cada rama puedes ser identificada por una etiqueta.

1.1 Notas preliminares.

- Para poder ejecutar el código de este capítulo es necesario haber ejecutado el código de los capítulos previos.
- Es necesario que la notebook se ejecute desde el directorio `demo`, ejecutando la siguiente celda.

[ ]:
```
cd demo
```

1.1.1 El archivo `.git/HEAD`.

Este archivo contiene la referencia a la rama en la que se encuentra un repositorio.

Ejemplo:

- la siguiente celda desplegará en contenido del archivo `.git/HEAD`, la cual corresponde a la rama con la etiqueta `master`.

[ ]:
```
cat .git/HEAD
```

[ ]:
```
tree .git
```

1.2 La rama `master`.

Al inicializar un repositorio, se define una sola rama. Dicha rama tienela etiqueta `master`.

1.3 El comando `git branch`.

El comando `git branch` cuenta con diversas opciones para la gestión de ramas. Este comando tiene diversas sintaxis, pero por lo pronto sólo se explorará la siguiente:

`git branch <opciones> <etiqueta>`

Donde: * `<etiqueta>` es el nombre de una etiqueta. * `<opciones>` es una combinación de una o más opciones. En caso de no tener opciones, el comando creará una rama nueva, asignándole la etiqueta definida por .

La referencia del comando `git branch` puede ser consultada en:

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

1.3.1 Opciones comunes de `git branch`.

- `--list`, la cual permite desplegar un listado de ramas. Incluso es posible definir patrones para el listado.
- `--delete` o `-d`, la cual permite eliminar una rama.

Ejemplo:

- La siguente celda creará una rama con la etiqueta `nueva_rama`.

[ ]:
```
git branch nueva-rama
```

- La siguiente celda listará todas las ramas del repositorio.

[ ]:
```
git branch -l
```

- La siguiente celda listará todas las ramas del repositorio que empiecen con `n`.

[ ]:
```
git branch --list "n*"
```

- La siguiente celda eliminará la rama con etiqueta `nueva-rama`

[ ]:
```
git branch -d nueva-rama
```

- La siguiente celda creará la rama con etiqueta `nueva`.

[ ]:
```
git branch nueva
```

[ ]:
```
git branch -l
```

[ ]:
```
git log
```

1.4 El comando `git checkout`.

El comando `git checkout` permite al usuario moverse entre ramas e incluso entre commits.

1.4.1 Cambio a una rama específica.

Para cambiar de una rama a otra se utiliza el comando `git checkout` con la siguiente sintaxis:

`git checkout <rama>`

Donde:

- `<rama>` corresponde a la etiqueta de la rama de destino.

Ejemplo:

- La siguiente celda moverá el repositorio a la rama `nueva`

[ ]:
```
git checkout nueva
```

[ ]:
```
git status
```

- Debido a que es una rama nueva, no se ven diferencias con respecto a la rama `master`.

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

- Esto se debe a que la rama `nueva` comparte el mismo commit.

[ ]:
```
git log
```

- Ahora se creará el archivo `archivo_nuevo`.

[ ]:
```
echo "Archivo de la rama nueva." > archivo_nuevo
```

[ ]:
```
cat archivo_nuevo
```

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

- Se añadirán todos los cambios al área de preparación.

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

- Se hará el primer commit en la rama `nueva`.

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

- Al ejecutar el comando `git log --oneline` el resultado será similar a los siguiente.

```
46c8014 (HEAD -> nueva) primer commit en nueva
ba397e8 (master) cuarto commit
cb0b6bb segundo commit
ed7d117 primer commit
```

Ahora el commit se ve reflejado en la rama `nueva`.

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

- La siguiente celda mostrará los archivos actuales.

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

- Ahora se moverá el repositorio a la rama `master`.

[ ]:
```
git checkout master
```

- El contenido es distinto.

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

- Se modificará al archivo `archivo-1`

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

[ ]:
```
echo "Otra linea" >> archivo-1
```

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

- Se hará un commit nuevo.

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

- Al ejecutar el comando `git log --oneline` el resultado será similar a los siguiente.

```
0834a0b (HEAD -> master) quinto commit
ba397e8 cuarto commit
cb0b6bb segundo commit
ed7d117 primer commit
```

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

1.5 Visualización de diferencias entre ramas.

El comando `git diff` puede visualizar las diferencias entre ramas de manera similar a como lo hace con los commits.

`git diff <etiqueta 1> <etiqueta 2>`

Donde:

- `<etiqueta 1>`y `<etiqueta 2>` son etiquetas de ramas.

Ejemplo:

- La siguiente celda desplegará las diferencias entre la rama `mastery` la rama `nueva`.

[ ]:
```
git diff master nueva
```

- Al igual que con los commits es posible especificar el archivo que se quiere observar.

[ ]:
```
git diff nueva master archivo-1
```

1.6 El comando `git merge`.

Este comando combina el contenido de una rama con otra.

La sintaxis es la siguiente:

```
git merge <rama>
```

Donde: * `<rama>` es la etiqueta de la rama que se fusionará con la actual.

El resultado es que los contenidos de ambas ramas serán combinados.

La referencia del comando `git merge` puede ser consultada en:

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

Es muy común que puedan ocurrir conflictos al realizar un merge, particularmente cuando dos desarrolladores editan el mismo archivo.

https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts

Ejemplo:

- A continuación se mezclarán los contenidos de la rama `nueva` en la rama actual (`master`).

Nota: El mensaje de error indica que la mezcla requiere de un paso adicional, debido a las restricciones de la notebook.

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

[ ]:
```
git merge nueva -m "commit fusionado"
```

[ ]:
```
git status
```

[ ]:
```
git log
```

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

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

[ ]:
```
cat archivo_nuevo
```

[ ]:
```
git diff nueva master
```

1.7 Restitución de un commit específico.

Es posible traer de vuelta el estado de un commit específico mediante el comando `git checkout`.

```
git checkout <identificador>
```

Donde:

- `<identificador>` corresponde a los primero dígitos del identificador del commit que se utilizará.

El resultado es que se restablecería el estado del commit referenciado, pero se encontraría en el estado `"Detached HEAD"`.

1.8 El estado "Detached HEAD".

Git permite acceder a un estado en el que el repositorio se encuentra desligado de cualquier rama y versión al que se denomina "Detached HEAD". Dicho estado permite realizar modificaciones, pero no estáran ligadas a alguna rama existente.

Ejemplo:

- Tomando en cuenta los siguientes commits en master:

```
f7ede1 (HEAD -> master) commit fusionado
0834a0b quinto commit
46c8014 (nueva) primer commit de nueva
ba397e8 cuarto commit
cb0b6bb segundo commit
ed7d117 primer commit
```

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

- Si se quisiera restaurar el estado del commit con descripción cuarto commit, se ejecutaría el siguiente comando:

```
git checkout ba397e8
```

Y el resultado será descrito con el siguiente mensaje:

```
Note: checking out 'ba397e8'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at ba397e8 cuarto commit
```

[ ]:
```
git checkout 0e768b7 #NOTA: Es necesario sustituir el número identificador.
```

[ ]:
```
git status
```

[ ]:
```
git log
```

- A partir del estado "Detached HEAD" se creará una nueva rama llamada restituida, la cual contendrá al estado restaurado.

[ ]:
```
git checkout -b restituida
```

[ ]:
```
git log
```

[ ]:
```
git branch -l
```

[ ]:
```
tree .git
```

[ ]:
```
git checkout master
```

[ ]:
```
ls -al
```