Tema 5 Sistemas de Control de Versiones

1. Más comandos de Git

Esta sección describe algunos de los comandos más interesantes de git (no tan conocidos).

1.1 Git stash (reserva)

La orden git stash nos permite salvar momentáneamente el espacio de trabajo cuando tenemos que cambiar de rama o preparar la rama actual para sincronizar cambios.

Las operaciones más importantes que podemos hacer con git stash son:

git stash save

Es equivalente a poner solo git stash pero nos permite realizar más acciones como:

```
git stash save "Tu mensaje"
```

```
git stash save -u
```

El parámetro -u permite que se almacenen también los ficheros sin seguimiento previo (*untracked* en inglés, aquellos ficheros que no se han metido nunca en el repositorio).

git stash list

Permite mostrar la pila del stash.

```
$ git stash list
```

```
stash@{0}: On master: Stash con mensaje
```

```
stash@{1}: WIP on master: 4ab21df First commit
```

git stash apply

Esta orden coge el stash que está arriba en la pila y lo aplica al espacio de trabajo actual. En este caso siempre es `stash@{0}`. El stash permanece en la pila.

Se puede indicar como parámetro un stash en concreto.

git stash pop

Funciona igual que git apply con la diferencia de que el stash sí se borra de la pila.

git stash show

Muestra un resumen de los ficheros que se han modificado en ese stash.

```
$ git stash show
A.txt | 1 +
B.txt | 3 +++
2 file changed, 4 insertions(+)
```

Para ver los cambios podemos usar el parámetro -p

```
$ git stash show -p
--- a/A.txt
+++ b/A.txt
@@ -45,6 +45,7 @@ nav:
+ This is a change
```

Por defecto siempre muestra la cabeza de la pila. Igual que en casos anteriores podemos indicar un stash en concreto.

```
$ git stash show stash@{1}
```

git stash branch

Permite crear una nueva rama a partir del último stash. Además, el mismo es borrado de la pila. Se puede especificar uno en concreto si lo queremos, como en el resto de comandos.

```
git stash branch nombre-de-nueva-rama stash@{1}
```

git stash clear

Este comando borrar todos los stash de la pila. Es destructiva y no se puede deshacer.

git stash drop

Permite borrar un stash en concreto (o el último si no se indica ninguno). Como con clear, borrarlo implica que no se puede recuperar.

1.2 Git worktree

Uno de los problemas más habituales es tener que tocar una rama distinta a la que tenemos actualmente. Eso implica que si estamos en medio de un trabajo tendríamos que hacer un commit o un stash, lo cual a veces es bastante molesto.

Con git worktree podemos crear un directorio de trabajo que contenga otra rama distinta, de forma temporal. No supone otro clon del repositorio porque ambos usan el mismo.

git worktree add

Esta función es la que crea el espacio de trabajo temporal. Imaginemos que estamos en una rama llamada develop:

```
$ git worktree add ../project-master master
```

```
$ git worktree add -b fix ../project-fix master
```

La primera orden crea un directorio llamado project-master que contiene el estado de master. La segunda, que contiene el parámetro -b equivale a crear una nueva rama llamada fix, que se crea desde master (suponemos que no existe fix).

git worktree list

Muestra el listado de directorios y espacios de trabajo.

```
$git worktree list

/home/sergio/taller-de-git  3b63b4b [master]

/home/sergio/fix           3b63b4b [fix]
```

git worktree remove

Borrar un espacio de trabajo. Hay que indicar el nombre entre corchetes que aparece en el listado

```
$ git worktree delete fix
```

git worktree prune

Una cuestión importante, es que las ramas que estén desplegadas en otro espacio de trabajo, se encuentran bloqueadas y no se pueden desbloquear en otro distinto.

Esto significa que si estamos trabajando en la rama developer, creamos otro worktree en otro directorio de la rama master, no podemos hacer pasar a master. No es posible tener la misma rama en varios espacios de trabajo.

Si se ha borrado el directorio a mano (en vez de usando remove), eso no implica que el bloqueo desparezca. Con esta orden podemos hacer que git compruebe que los espacios de trabajo secundario se comprueben de nuevo para ver si siguen existiendo y se elimine el bloqueo.

1.3 Git blame

Lo ideal en un equipo de desarrollo es que el código pase por todas las manos para así mejorar su calidad.

Con git blame podemos saber quién fue el último en modificar una línea concreta de código, en qué commit y en qué fecha lo hizo.

```
$ git blame ejemplo.php

33cdd02c (Sergio Gómez 2020-01-20 16:58:52 +0100  8)   name: "material"

33cdd02c (Sergio Gómez 2020-01-20 16:58:52 +0100  9)   language: "es"
```