26_git_stash

May 6, 2020

1 Preservación de trabajo en progreso.

Es común que sea necesario "pausar" el trabajo que se realiza en un repositorio y restituir un directorio de trabajo "limpio", pero dicho trabajo aún no amerita realizar un "commit".

Git permite definir unestado conocido como "trabajo en proceso" (*WIP* por sus siglads en inglés).

1.1 Preliminares:

A fin de contar con un entorno unificado, se utilizará una versión creada previamente del directorio `demo` que incluye los ejercicios de los capítulos previos y se encuentra comprimida en el archivo `src/prueba.zip`.

[1]:
```
rm -rf prueba
```

[2]:
```
unzip src/prueba.zip
```

```
Archive:  src/prueba.zip
   creating: prueba/
 extracting: prueba/archivo-2
 extracting: prueba/archivo-1
   creating: prueba/.git/
  inflating: prueba/.git/index
   creating: prueba/.git/refs/
   creating: prueba/.git/refs/heads/
  inflating: prueba/.git/refs/heads/master
 extracting: prueba/.git/refs/heads/restituida
 extracting: prueba/.git/refs/heads/nueva
   creating: prueba/.git/refs/tags/
  inflating: prueba/.git/description
   creating: prueba/.git/info/
  inflating: prueba/.git/info/exclude
 extracting: prueba/.git/ORIG_HEAD
 extracting: prueba/.git/COMMIT_EDITMSG
   creating: prueba/.git/objects/
   creating: prueba/.git/objects/90/
 extracting: prueba/.git/objects/90/a8b80bbb845cb1187852777769937a9574e3ab
   creating: prueba/.git/objects/ed/
 extracting: prueba/.git/objects/ed/7d117fc8970b297653c23ebc41936cc4ab4472
```

```
   creating: prueba/.git/objects/pack/
   creating: prueba/.git/objects/cb/
extracting: prueba/.git/objects/cb/0b6bb3a84bc6fcd0f6a0f87909bf68f298e21a
   creating: prueba/.git/objects/2c/
extracting: prueba/.git/objects/2c/b76ef80c41c0037c6a82092737cba6061c7083
   creating: prueba/.git/objects/info/
   creating: prueba/.git/objects/e6/
extracting: prueba/.git/objects/e6/9de29bb2d1d6434b8b29ae775ad8c2e48c5391
   creating: prueba/.git/objects/a1/
extracting: prueba/.git/objects/a1/9abfea0f29b668c91c58c834b8965e6c37804f
   creating: prueba/.git/objects/dc/
extracting: prueba/.git/objects/dc/d3ac8c60d63186964b45f87cc4416e6caefa5f
   creating: prueba/.git/objects/78/
extracting: prueba/.git/objects/78/8564428c6076630b22efc89ad20c5f63ee9bf1
   creating: prueba/.git/objects/ba/
extracting: prueba/.git/objects/ba/397e8f7c51f8dfdefa12250933811042df91a2
   creating: prueba/.git/objects/ff/
extracting: prueba/.git/objects/ff/7ede1f3ce0201ae532684bbee157247c3a88b7
   creating: prueba/.git/objects/ac/
extracting: prueba/.git/objects/ac/9103394fb02ee49adb4c05e2125174707fbb7d
   creating: prueba/.git/objects/1f/
extracting: prueba/.git/objects/1f/1f5adbb5e6ff2e2ff651e352f21df0417defb2
   creating: prueba/.git/objects/08/
extracting: prueba/.git/objects/08/34a0b252e02bc28bae34588e01f410ad771884
   creating: prueba/.git/objects/d7/
extracting: prueba/.git/objects/d7/f0e7ba5cf079cdea1d7c8792e1432161f082c1
   creating: prueba/.git/objects/46/
extracting: prueba/.git/objects/46/c8014e29a665bd0e883c9008ad4412f8933809
   creating: prueba/.git/objects/df/
extracting: prueba/.git/objects/df/8a330f084618bdb823d99a6997d7be4b9efd15
   creating: prueba/.git/objects/b3/
extracting: prueba/.git/objects/b3/890082fec489a5ecff76db4fd36cc10d4bc8d6
   creating: prueba/.git/branches/
   creating: prueba/.git/logs/
   creating: prueba/.git/logs/refs/
   creating: prueba/.git/logs/refs/heads/
  inflating: prueba/.git/logs/refs/heads/master
  inflating: prueba/.git/logs/refs/heads/restituida
  inflating: prueba/.git/logs/refs/heads/nueva
  inflating: prueba/.git/logs/HEAD
extracting: prueba/.git/HEAD
  inflating: prueba/.git/config
   creating: prueba/.git/hooks/
  inflating: prueba/.git/hooks/post-update.sample
  inflating: prueba/.git/hooks/commit-msg.sample
  inflating: prueba/.git/hooks/pre-receive.sample
  inflating: prueba/.git/hooks/update.sample
  inflating: prueba/.git/hooks/pre-push.sample
```

```
  inflating: prueba/.git/hooks/fsmonitor-watchman.sample
  inflating: prueba/.git/hooks/prepare-commit-msg.sample
  inflating: prueba/.git/hooks/pre-applypatch.sample
  inflating: prueba/.git/hooks/pre-rebase.sample
  inflating: prueba/.git/hooks/applypatch-msg.sample
  inflating: prueba/.git/hooks/pre-commit.sample
 extracting: prueba/archivo_nuevo
 extracting: prueba/invisible
 extracting: prueba/.gitignore
```

[3]:
```
cd prueba
```

[4]:
```
git branch
```

```
* master
  nueva
  restituida
```

[5]:
```
git log --oneline
```

```
ff7ede1 (HEAD -> master) commit
fusionado
0834a0b quinto commit
46c8014 (nueva) primer commit de la rama nueva
ba397e8 (restituida) cuarto commit
cb0b6bb segundo commit
ed7d117 primer commit
```

1.2 El comando `git stash`.

Este comnando permite preservar tanto el directorio de trabajo como el índice actual y regresar a `HEAD` sin necesidad de realizar un commit.

`git stash <opciones y argumentos>`

Ejecutar `git stash` sin opciones es queivalente a `git stash push`.

Para mayor referencia es posible consultar la siguiente liga:

https://www.git-scm.com/docs/git-stash

1.2.1 El comando `git stash push`.

Este comando realiza las siguientes acciones:

- Crea una referencia guardada en el archivo `.git/refs/stash` del repositorio actual.
- Guarda el estado del directorio de trabajo y del índice del repositorio.
- Regresa el estado del repositorio a `HEAD`.

Nota: Es posible guardar más de in estado *WIP* usando `git stash push`.

Ejemplo:

- La siguiente celda creará al archivo `preservado-2` .

[6]:
```
touch preservado-1
```

- La siguiente celda modificará a `archivo-2`.

[7]:
```
echo "nueva línea" >> archivo-2
```

[8]:
```
cat archivo-2
```

```
nueva línea
```

[9]:
```
ls
```

```
archivo-1  archivo-2  archivo_nuevo  invisible  preservado-1
```

La siguiente celda mostrará las modificaciones del repositorio con respecto a `HEAD`.

[10]:
```
git status
```

```
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>…" para actualizar lo que será confirmado)
  (usa "git checkout -- <archivo>…" para descartar los cambios en el
directorio de trabajo)

        modificado:     archivo-2

Archivos sin seguimiento:
  (usa "git add <archivo>…" para incluirlo a lo que se será confirmado)

        preservado-1

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

- Se añadirán las modificaciones al índice.

[11]:
```
git add --all
```

- Se ejecutará el comando`git stash push`.

[12]:
```
git stash push
```

```
Directorio de trabajo guardado y estado de índice WIP on master: ff7ede1 commit
fusionado
```

- El sistema de archivos ha sido restablecido a `HEAD`.

[13]:
```
ls
```

```
archivo-1  archivo-2  archivo_nuevo  invisible
```

[14]:
```
git status
```

```
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
```

[15]:
```
git log --oneline
```

```
ff7ede1 (HEAD -> master) commit
fusionado
0834a0b quinto commit
46c8014 (nueva) primer commit de la rama nueva
ba397e8 (restituida) cuarto commit
cb0b6bb segundo commit
ed7d117 primer commit
```

- La siguiente linea mostrará la estructura del directorio `.git/refs` del repositorio. El resultado es algo similar a:

```
.git/refs
  heads
      master
      nueva
      restituida
  stash
  tags
```

[16]:
```
tree .git/refs
```

```
.git/refs
  heads
      master
      nueva
      restituida
  stash
  tags

2 directories, 4 files
```

- El contenido de `.git/refs/stash` es un identificador.

[17]:
```
cat .git/refs/stash
```

```
e3d8a0db6907ddfa0efa87b908832f8bb2f3f2f1
```

- La siguiente celda creará al archivo `preservado-2`.

[18]:
```
echo Más contenido > preservado-2
```

- Se ejecutará el comando `git stash`.

[19]:
```
git add --all
```

[20]:
```
git stash
```

```
Directorio de trabajo guardado y estado de índice WIP on master: ff7ede1 commit
fusionado
```

1.2.2 El comando `git stash list`.

Este comando regresa un listado de estados *WIP* de un repositorio. Cada línea se describe de la siguiente manera:

`stash@{<n>} WIP on <rama>: <identificador> <mensaje>`

Donde:

- `<n>` es un número que comienza en `0` y va aumentando de uno en uno. El número `0` corresponde al evento más reciente.
- `<rama>` corresponde a la rama en la que se realizó el "stash".
- `<identificador>` corresponde al identificador del commit en el que se realizó el "stash".
- `<mensaje>` corresponde al mensaje del commit en el que se realizó el "stash".

Ejemplo:

La siguiente celda contiene la lista de los "stash" almacenados.

[21]:
```
git stash list
```

```
stash@{0}: WIP on master: ff7ede1 commit fusionado
stash@{1}: WIP on master: ff7ede1 commit fusionado
```

1.2.3 El comando `git stash show`.

Este comando muestra las modificaciones al estado de un stash.

`git stash show <stash>`

Donde:

- `<stash>` es un stash específico usando la sintaxis `stash@{<n>}`. El valor por defecto es `stash@{0}`.

Ejemplos:

[22]:
```
git stash show
```

```
 preservado-2 | 1 +
 1 file changed, 1 insertion(+)
```

[23]:
```
git stash show stash@{0}
```

```
 preservado-2 | 1 +
 1 file changed, 1 insertion(+)
```

[24]:
```
git stash show stash@{1}
```

```
 archivo-2    | 1 +
 preservado-1 | 0
 2 files changed, 1 insertion(+)
```

1.2.4 El comando `git stash branch`.

Este comando crea una nueva rama que será una bifurcación de la rama desde la que se originó el stash indicado y la que se le volcará el contenido del stash y éste será elimiando.

`git stash <rama> <stash>`

Donde:

- `<rama>` es la etiqueta de la nueva rama.
- `<stash>` la referencia a un stash en particular. El valor por defecto es `stash${0}`.

Ejemplo:

- La siguiente celda creará a la ramam `en_progreso` usando a `stash@{1}`.

[25]:
```
git stash branch en_progreso stash@{1}
```

```
Cambiado a nueva rama 'en_progreso'
En la rama en_progreso
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>…" para sacar del área de stage)

        modificado:     archivo-2
        nuevo archivo:  preservado-1

Botado stash@{1} (e3d8a0db6907ddfa0efa87b908832f8bb2f3f2f1)
```

[26]:
```
git branch
```

```
* en_progreso
  master
  nueva
  restituida
```

[27]:
```
git log --oneline
```

```
ff7ede1 (HEAD -> en_progreso,
master) commit fusionado
0834a0b quinto commit
46c8014 (nueva) primer commit de la rama nueva
ba397e8 (restituida) cuarto commit
cb0b6bb segundo commit
ed7d117 primer commit
```

[28]:
```
git status
```

```
En la rama en_progreso
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>…" para sacar del área de stage)

        modificado:     archivo-2
        nuevo archivo:  preservado-1
```

[29]:
```
ls
```

```
archivo-1  archivo-2  archivo_nuevo  invisible  preservado-1
```

[30]:
```
git stash list
```

```
stash@{0}: WIP on master: ff7ede1 commit fusionado
```

- Se realizará un commit con todas las modificaciones.

[31]:
```
git add --all
```

[32]:
```
git commit -m "trabajo en firme"
```

```
[en_progreso f5d98b8] trabajo en firme
 2 files changed, 1 insertion(+)
 create mode 100644 preservado-1
```

1.2.5 El comando `git stash apply`.

Este comando aplica un stash específico a una rama que no necesariamente es desde la que se originó el estas. Dicho stash no es eliminado al aplicarse.

`git apply <stash>`

Donde:

- `<stash>` la referencia a un stash en particular. El valor por defecto es `stash${0}`.

Ejemplo:

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

[33]:
```
git checkout nueva
```

```
Cambiado a rama 'nueva'
```

[34]:
```
ls
```

```
archivo-1  archivo-2  archivo_nuevo  invisible
```

- Se ejecutará el comando `git stash apply`.

[35]:
```
git stash apply
```

```
En la rama nueva
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>…" para sacar del área de stage)

        nuevo archivo:  preservado-2
```

- El stash no fue eliminado.

[36]:
```
git stash list
```

```
stash@{0}: WIP on master: ff7ede1 commit fusionado
```

1.2.6 El comando `git stash pop`.

Este comando permite volcar un stash en una rama. Una vez aplicado, el stash será eliminado.

`git stash pop <stash>`

Donde:

- `<stash>` la referencia a un stash en particular. El valor por defecto es `stash${0}`.

Ejemplo:

- Se ejecutará el comando `stash pop`.

[37]:
```
git stash pop
```

```
En la rama nueva
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>…" para sacar del área de stage)

        nuevo archivo:  preservado-2

Botado refs/stash@{0} (d4995b7d6da6de4b62098aa3e570f7c208db1112)
```

[38]:
```
ls
```

```
archivo-1  archivo-2  archivo_nuevo  invisible  preservado-2
```

- La siguiente celda mostrará el listado de stash vacío.

[39]:
```
git stash list
```

1.2.7 El comando `git stash drop`.

Este comando eliminará a un stash del listado.

`git stash pop <stash>`

Donde:

- `<stash>` la referencia a un stash en particular..El valor por defecto es `stash${0}`.

Ejemplo:

- La siguiente celda moverá al repositorio a la rama `master`.

[40]:
```
git checkout master
```

```
A       preservado-2
Cambiado a rama 'master'
```

- La siguiente celda creará al archivo `preservado-3`.

[41]:
```
echo "Nuevo archivo." > preservado-3
```

- Las siguientes celdas crearán un nuevo stash.

[42]:
```
git add --all
```

[43]:
```
git stash
```

```
Directorio de trabajo guardado y estado de índice WIP on master: ff7ede1 commit
fusionado
```

[44]:
```
git stash list
```

```
stash@{0}: WIP on master: ff7ede1 commit fusionado
```

- La siguiente celda creará al archivo `preservado-4`.

[45]:
```
echo "Archivo nuevo." > preservado-4
```

- Las siguientes celdas crearán un nuevo stash.

[46]:
```
git add --all
```

[47]:
```
git stash
```

```
Directorio de trabajo guardado y estado de índice WIP on master: ff7ede1 commit
fusionado
```

[48]:
```
git stash list
```

```
stash@{0}: WIP on master: ff7ede1 commit fusionado
stash@{1}: WIP on master: ff7ede1 commit fusionado
```

- La siguiente celda eliminará a `stash@{1}` del listado de stash.

[49]:
```
git stash drop stash@{1}
```

```
Botado stash@{1} (405216bd17fb3b2dd60846361685bbefac999772)
```

[50]:
```
git stash list
```

```
stash@{0}: WIP on master: ff7ede1 commit fusionado
```

1.2.8 El comando `git stash clear`.

Este comando limpia el listado de stash.

`git stash clean`

Ejemplo:

- La siguiente celda limpiará el listado de stash.

[51]:
```
git stash clear
```

[52]:
```
git stash list
```