# Academia Java | Xideral – Accenture

Eduardo Aguilar López

***Git & Github***

***Stach***

El comando git stash almacena temporalmente (o guarda en un stash) los cambios que hayas efectuado en el código en el que estás trabajando para que puedas trabajar en otra cosa y, más tarde, regresar y volver a aplicar los cambios más tarde. Guardar los cambios en stashes resulta práctico si tienes que cambiar rápidamente de contexto y ponerte con otra cosa, pero estás en medio de un cambio en el código y no lo tienes todo listo para confirmar los cambios.

El comando `git stash` coge los cambios sin confirmar (tanto los que están preparados como los que no), los guarda aparte para usarlos más adelante y, acto seguido, los deshace en el código en el que estás trabajando. Por ejemplo:

```
$ git status
On branch main
Changes to be committed:

    new file:   style.css

Changes not staged for commit:

    modified:   index.html

$ git stash
Saved working directory and index state WIP on main: 5002d47 our new homepage
HEAD is now at 5002d47 our new homepage

$ git status
On branch main
nothing to commit, working tree clean
```

Llegados a este punto, tienes libertad para hacer cambios, crear confirmaciones, cambiar de rama y efectuar cualesquiera otras operaciones de Git; y, luego, regresar y volver a aplicar el stash cuando lo tengas todo listo.

Ten en cuenta que el stash es local para tu repositorio de Git y que los stashes no se transfieren al servidor cuando subes los cambios al repositorio remoto.

*Como volver a aplicar los cambios de un stach*

Puedes volver a aplicar los cambios de un stash mediante el comando `git stash pop`:

```
$ git status
On branch main
```

```
nothing to commit, working tree clean
$ git stash pop
On branch main
Changes to be committed:

    new file:   style.css

Changes not staged for commit:

    modified:   index.html

Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)
```

Al hacer pop del stash, se eliminan los cambios de este y se vuelven a aplicar en el código en el que estás trabajando.

***Git clean***

Mientras estamos trabajando en un repositorio podemos añadir archivos a él, que realmente no forma parte de nuestro directorio de trabajo, archivos que no se deberían de agregar al repositorio remoto.

El comando clean actúa en archivos sin seguimiento, este tipo de archivos son aquellos que se encuentran en el directorio de trabajo, pero que aún no se han añadido al índice de seguimiento de repositorio con el comando add.

```
$ git clean
```

La ejecución del comando predeterminado puede producir un error. La configuración global de Git obliga a usar la opción `force` con el comando para que sea efectivo. Se trata de un importante mecanismo de seguridad ya que este comando no se puede deshacer.

*Revisar los archivos que no tiene seguimiento*

```
$ git clean –dry-run
```

Eliminar los archivos listados de no seguimiento

```
$ git clean -f
```

***Git cherry-pick***

Git chery-pick se define como un importante comando del sistema de Git que se encarga de elegir uno o más commit o confirmaciones de cambios de una rama específica para luego aplicarla a otra rama.

Esto quiere decir que los commit de la plataforma de Git pueden elegirse por referencia para añadirlos al HEAD actual de trabajo, que no es más que el commit en el que se encuentra posicionado el repositorio del usuario en cada momento.

*Características de git Cherry-pick*

Dentro de las propiedades y características del comando cherry-pick, se encuentra que es de fácil ejecución, hasta el punto en el que, para utilizarlo, solo se necesita conocer el commit determinado que se desea aplicar en la rama.

Para el uso de este comando también es importante que no se hayan llevado a cabo modificaciones del commit de HEAD, para poder tener un espacio de trabajo limpio.

El Git cherry-pick también se caracteriza por su utilidad para la rápida corrección de errores en el sistema, evitando que impacten sobre más usuarios. Al mismo tiempo, esta herramienta puede utilizarse para el trabajo independiente en un mismo proyecto de desarrollo, gracias a que permite ir avanzando en los procesos de manera sencilla y práctica.\

Git cherry-pick ofrece múltiples opciones que extienden sus funciones, como, por ejemplo:

- *-e* o *--edit*: solicita un mensaje de *commit* o confirmación antes de llevar a cabo la ejecución del comando.
- *--no-commit*: en vez de realizar una confirmación nueva, mueve el contenido al directorio de trabajo de la rama actual.

*Limitaciones*

Git cherry-pick también se caracteriza por su gran potencia de ejecución, por lo que su uso no es siempre recomendable y debe ser cuidadoso para evitar inconvenientes,
como commits duplicados. Además, aunque la ejecución de este comando se realizara de forma exitosa, en algunos casos es preferible trabajar con fusiones tradicionales de Git.

***Bibliografía:***


- https://www.atlassian.com/es/git/tutorials/saving-changes/git-stash
- https://platzi.com/clases/1557-git-github/19983-git-clean-limpiar-tu-proyecto-de-archivos-no-desea/
- https://keepcoding.io/blog/que-es-git-cherry-pick/