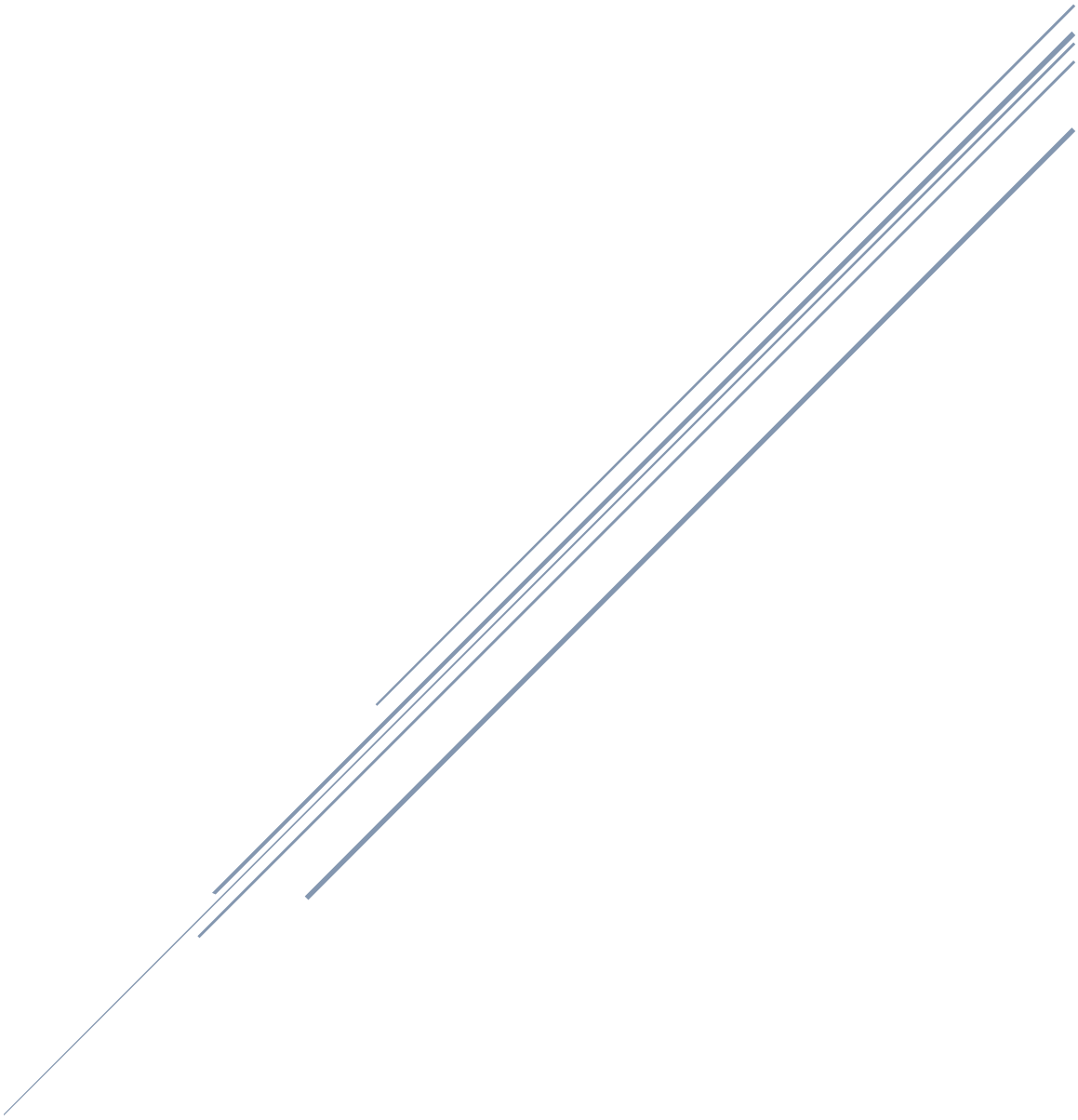
GIT / GITHUB

Manual para Windows



Recopilación extraída del curso de Git y GitHub de Platzi. Todos los créditos corresponden a Freddy Vega. Enlace del curso: https://platzi.com/cursos/git-github/

Introducción

Los sistemas de control de versiones guardan cambios los atómicos de un archivo y dejan un registro de quien, como y cuando se hicieron.

GIT

Git precisamente es un sistema de control de versiones que guarda los cambios atómicos de un archivo. Comandos iniciales:

$git init = Permite inicializar un repositorio en un fichero especifico
$git add file.dom = El archivo se agrega a "staging"
$git commit -m"comments" = Se envían los cambios a la base de datos (repositorio)

Caso:

```
1. ***CAMBIOS A FILE.TXT***
2. $git add file.txt
3. $git commit -m"Primer commit"
```

Comandos iniciales

$git status = Muestra el estado del repositor (archivos por agregar o por hacer commit)
$git log file.dom = Muestra todos los cambios hechos a un archivo.

Git cuenta con tres estados para archivos: Staged, Modified y Commited. Git analiza a precisión archivos de texto sin embargo Git pierde precisión analizando archivos binarios.

Comandos básicos de Linux

$pwd =Muestra nuestra ubicación con respecto al sistema de archivos
$cd =Moverse a otro directorio
$ls =Listar archivos
$ls -al =Listar todos los archivos, incluso los ocultos
$clear =Limpiar consola
$cd .. =Regresar al directorio anterior
$mkdir "nameFolder" = Crear directorio
$touch "nameFile" = Crear archivo
$cat "nameFile" = Muestra contenido del archivo
$history = Muestra todos los comandos ejecutados
$rm "fileName" = Borrar archivo
$cualquierComando –help = Da una descripción del comando

Comandos básicos de Git

Iniciando repositorio

1. Nos aseguramos de estar posicionados en el directorio de trabajo
2. $git init
3. ***AGREGAR / MODIFICAR ARCHIVO***
4. $git commit -m”comments”

Configurar variables de entorno (antes del primer commit)

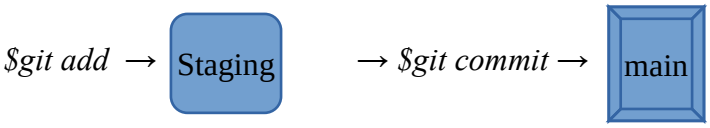
1. $git config --global user.name “First Name”
2. $git config --global user.email “email@domain.com”

Analizar cambios en archivos

$git show = Muestra que y en donde se hicieron cambios
$git log file.dom = Muestra información de los commit por archivo
$git log –stat = Muestra que líneas fueron modificadas o agregadas
$git diff commitCode1 commitCode2 = Muestra diferencias entre dos versiones

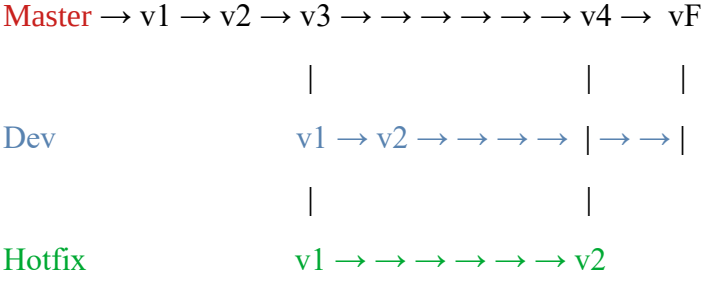
Staging

Cuando *$git init* se ejecuta en un directorio se crea el staging y el repositorio (main). Staging es el estado temporal antes de enviar los cambios al repositorio.



Branch (Ramas)

Las ramas son entidades que almacenan cambios especificos de un archivo. Las ramas tienden a separarse o integrarse a la rama principal (main). La diferencia entre main y las demás ramas es que se considera que la rama main es quien está en producción.



Volver en el tiempo a una versión antigua

$git reset commitCode –hard = Regresar en definitiva sin dejar el staging disponible
$git reset commitCode –soft = Regresaren definitiva dejando el staging disponible

Traer de vuelta una versión anterior

$git checkout commitCode file.dom = Muestra como era el archivo en ese commit

**En caso de querer guardar los cambios de versión anterior:

```
1. $git checkout commitCode file.dom
2. $git add .
3. $git commit -m“comments”
```

Git reset vs Git rm

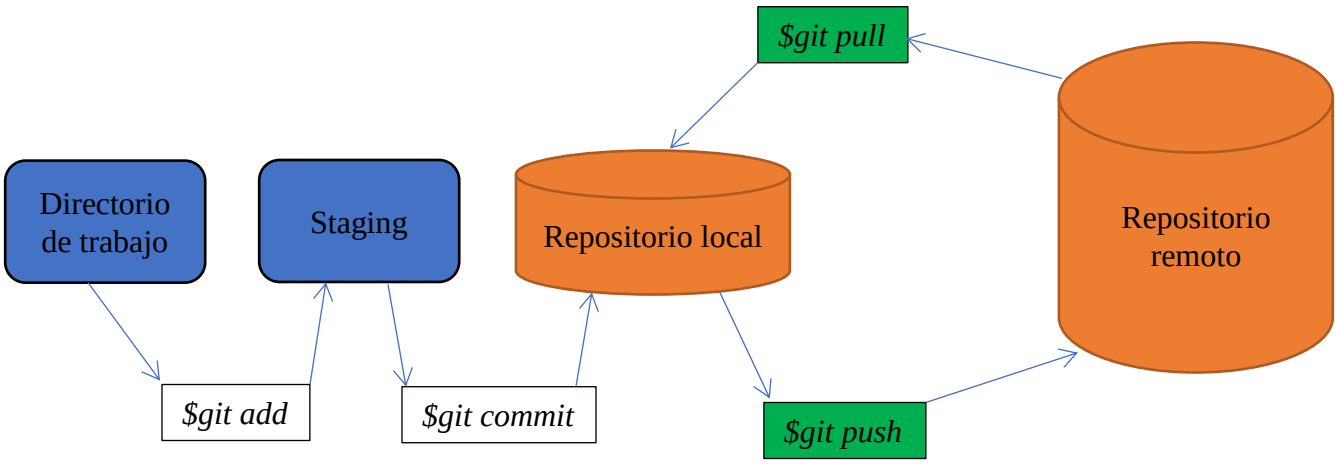
$git rm

- *$git rm --cached* = Elimina los archivos del repositorio local y del área de staging. Básicamente se dejan de trackear los archivos.
- *$git rem --force* = Elimina los archivos de Git y el disco duro

$git reset

- *$git reset --soft* = Se borra todo el historial y los registros de Git, sin embargo, se guardan los cambios que estén alojados en Staging (para posteriormente hacer commit si se desea)
- $git rem --force = Elimina los archivos de Git y el disco duro

Flujo de trabajo básico de Git



Introducción a las ramas

$git branch "nombreRama" = Crear una nueva rama
$git checkout "nombreRama" = Moverse a una rama

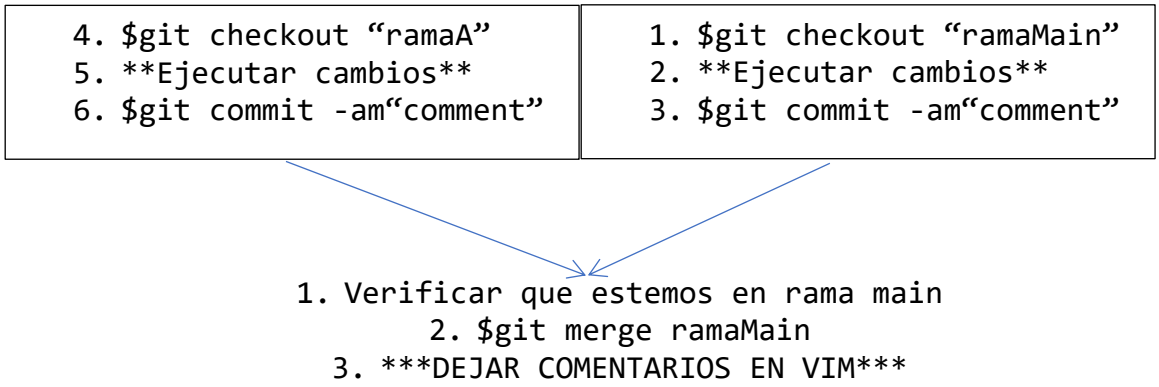
Caso: Hacer cambios a un archivo en una rama

1. `$git checkout "rama"`
2. `***EJECUTAR CAMBIO EN RAMA***`
3. `$git add file.dom`
4. `$git commit -m"comments"`

Fusión de ramas con merge

Un merge a producción siempre se tiene que hacer sobre la rama main.

Ejemplo de un merge:



Conflicto en merge

En caso de que un surja un conflicto durante el merge, hay que asegurarnos de que estemos en la rama main. Resolución:

1. ***RESOLVER CONFLICTO (En VSC u otro editor de texto)***
2. ***GUARDAR ARCHIVO***
3. *$git commit -am "comments"*

*El comando *$git commit -am "comments"* nos permite agregar el archivo (*$git add*) y hacer un commit (*$git commit*) al mismo tiempo. Este comando solo se puede ejecutar si el archivo ya fue agregado a staging previamente (*$git add*)

Trabajando con repositorios remotos

Configuración básica para trabajar con un repositorio remoto. Antes de ejecutar algún comando para importar un repositorio nos tenemos que asegurar que

1. Importar un repositorio remoto

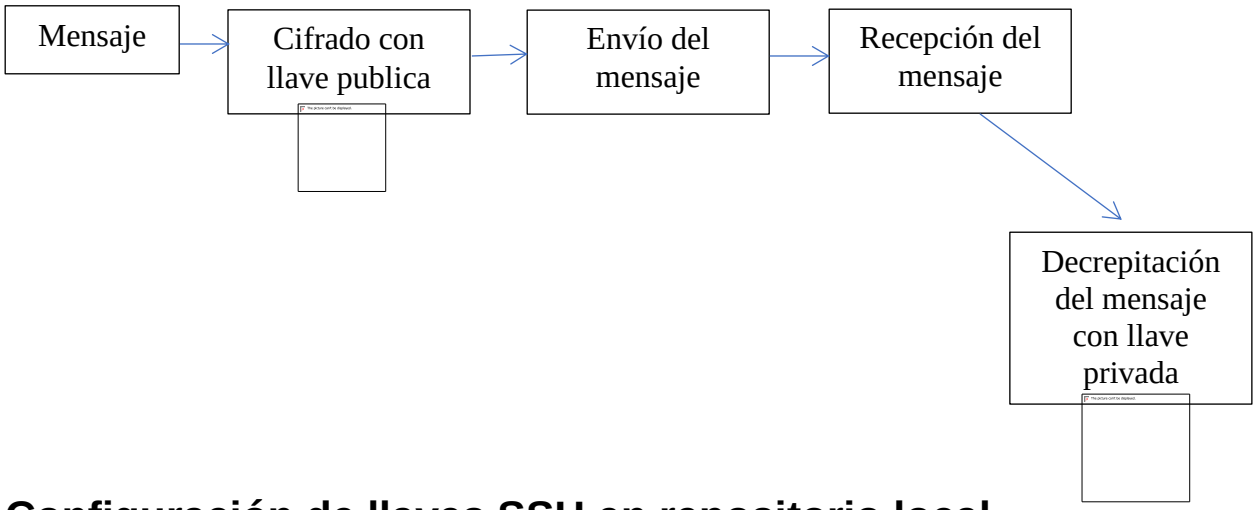
   $git remote add origin <https> = Referencia al repositorio remoto
   $git remote -v = Verificación de agregación del repositorio remoto
   $git pull origin main --allow-unrelated-histories = Por defecto, el comando git merge (el primer pull) rechaza fusionar historias que no comparten un ancestro común. Esta opción puede usarse para anular esta seguridad cuando se fusionan historias de dos proyectos que empezaron sus vidas de forma independiente.

2. Exportar un repositorio local

   $git pull origin main = Se envía nuestro repositorio local al repositorio remoto

Llaves públicas y privadas

Funcionamiento del protocolo:



Configuración de llaves SSH en repositorio local

1. En consola moverse a home con *cd*
2. $*ssh -keygen -t rsa -b 4096 -c"userGitEmail@email.com"*
3. $*eval $(ssh -agent -s)* = Se evalúa si el servidor ssh está activo
4. $*ssh-add ~/.ssh/id_rsa* = Se agrega la llave al servidor

Conexión a GitHub con SSH

1. (En GitHub) Ir a settings >> Seleccionar **SSH and GPT Keys**
2. Seleccionar **New SSH key**
3. Pegar contenido del archivo de la llave publica >> Seleccionar **Add SSH Key**
4. (En Git) *$git remote set set-url origin <SSH link>*

Tag y versiones en Git

Los tags se utilizan típicamente para señalar la culminación de una versión. Proceso para agregar tags:

1. $*git log --all --graph --decorate --oneline* = Muestra las ramas graficamente
2. $*git tag -a tagName -m"comments" commitCode* = Se crea el tag
3. $*git tag -l* = Ver los tags creados
4. $*git pull origin main*
5. $*git push origin --tag* = Se envía el tag a GH

Borrar tag local: *$git tag -d tagName*

Borrar tag remote: *$git push --delete origin tagName*

Manejo de ramas en GitHub

$git show -branch -all = Muestra detalles e historia de las ramas

$git k = Muestra interfaz visual de la historia de las ramas

$git push origin branchName = Enviar ramas (que no son main) a GitHub

Merge de ramas de desarrollo a main

Las imágenes al ser archivos binarios no deberían ser agregadas al repositorio. La razón es por la transmisión constante de datos que hacen al hacer *$git pull / $git push*, las imágenes al ser elementos mas pesados atrasan la ejecución de estos comandos.

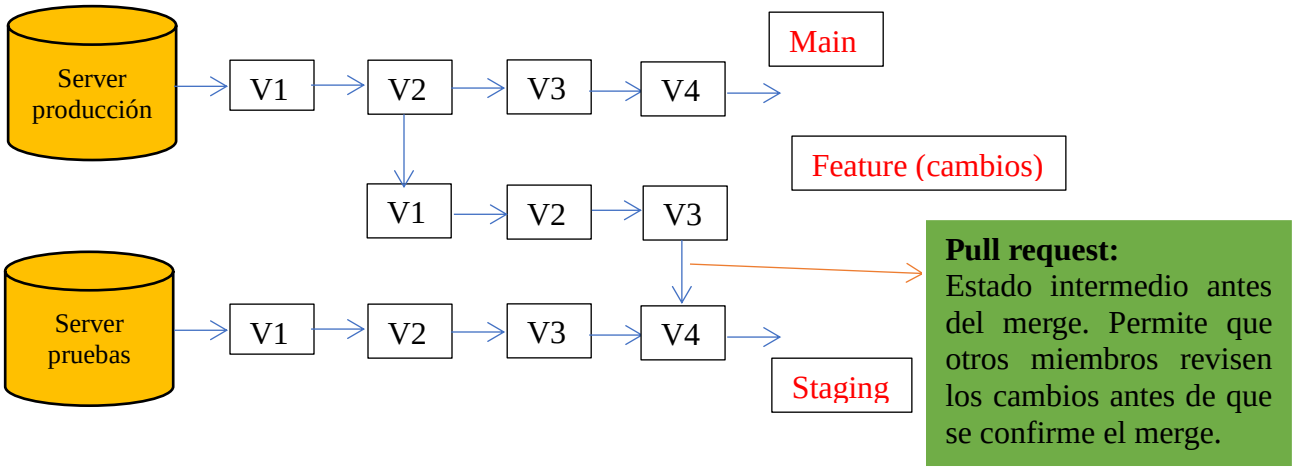
Merge:

1. ***HACER CAMBIO EN RAMOS***
2. $git commit -am"comments"
3. $git checkout main
4. $git merge branch1
5. $git merge branch2

6. `$git pull origin main`
7. `$git push origin main`

Flujo de trabajo profesional con Pull request

En un entorno de trabajo profesional se bloquea la rama main, solo se puede hacer merge en main si hay un code review. Como alternativa se crea una rama "espejo" a main llamada staging. La rama staging tiende a emular el entorno de la rama main. Todos los cambios se prueban de primero en la rama staging.



Utilizando un pull request en GitHub

1. `***Hacer cambios el la rama feature / dev***`
2. `$git pull origin feature`
3. `$git push origin feature`
4. `En GH: Pull Request >> New Pull Request >> Compare <main> con <feature>`

Para quien aprueba (GH):

1. Ir a Pull Requests >> Seleccionar P.R
2. Seleccionar files changed >> Seleccionar Review Changes
   - Opcion1 -> Enviar retroalimentacion
   - Opcion 2 -> Aprobar el Pull Request

Para quien desarrolla (Si el cambio no se aprueba):

1. `$git pull origin feature`
2. `***Hacer cambio en archivo***`
3. `$git commit -am"comments"`
4. `$git pull origin feature`

5. `$git push origin feature`

Para quien aprueba (GH):

1. Ir a Pull Requests >> Seleccionar Files Changed >> Review changes >> Aprobar cambio
2. Ir a Pull Requests >> Seleccionar Conversation >> Merge
3. Ir a Pull Requests >> Seleccionar Conversation >> Borrar Branch (opcional)

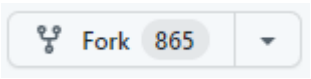
Para quien desarrolla (Una vez se aprueba el cambio):

1. `$git checkout main`
2. `$git pull origin main ->` Para actualizar repositorio local

Creando un Fork

Fork es como podemos contribuir a un repositorio Open Source. Hacer un fork es clonar el estado de un proyecto.

1. Ir al proyecto y hacer fork



2. Una vez copiar, copiar el link HTTPS
3. En consola: *$git clone <https link>*
4. ***HACER CAMBIOS EN ARCHIVO***
5. $*git push*
6. En GitHub: Pull Requests >> Select New Pull Request
7. ***ESPERAR A QUE EL AUTO AUTORICE / DESAPRUEBE EL P.R***

Para quien aprueba:

1. Se recibe notificación del Pull Request
2. Ingresar al Pull Request
3. Seleccionar Files Changed >> Review Changes >> Approve / Disapprove
4. Merge Pull Request

Ignorando archivos en el repositorio con .gitignore

Algunas veces es necesario ignorar archivos (especialmente imágenes, passwords, API codes, etc.), para ignorar estos archivos (para que no se envíen a GH) creamos un archivo .gitignore

1. En el directorio de trabajo, crear el archivo .gitignore

2. ***Escribir en el archivo: *.jpg*** -> Para ignorar imágenes.
3. $*git commit -am"comments"*
4. $*git pull origin main*
5. $*git push origin main*

Múltiples entornos de trabajo en Git

Git Rebase: Reorganizando el trabajo realizado

Un rebase es básicamente tomar una rama entera y pegarla por completo a otra rama (en lugar de hacer merge). Rebase es solo para repositorios locales, nunca se debería de enviar a un repositorio remoto. Hacer rebase no es una buena practica porque se pierde el origen de las ramas.

Suponiendo que queremos integrar la rama branch1 a main:

1. $*git checkout branch1*
2. ****HACER CAMBIOS****
3. $*git rebase main*
4. *$git checkout main*
5. *$git rebase branch1*

Git Stash: Guardar cambios en memoria y recuperarlos después

Supongamos que estamos e una rama y hacemos modificaciones al archivo, sin embargo no podemos hacer *$git add* o *$git commit* por que necesitamos ir a otra rama a verificar información o por cualquier otro motivo. Podemos deducir que necesitamos alguna herramienta que nos permita guardar cambios temporalmente para recuperarlos después. Para ello usamos *$git stash*, proceso:

1. ***HACEMOS CAMBIOS EN RAMA1***
2. *$git stash* = Almacenamiento temporal
3. *$git stash list* = Muestra el código de los WIP (Work in Progress)
4. *$git checkout rama2* = Consultamos información que requiramos
5. *$git checkout rama1*
6. *$git stash pop* = Volvemos a visualizar los cambios que habíamos hecho
7. *$git commit -am"comments"*

Mandar cambios temporales en una rama

1. ***HACEMOS CAMBIOS EN RAMA1***
2. *$git stash*

3. *$git stash branch nombreRamaCambios*
4. (nos movemos automáticamente a ramaCambios)
5. *$git commit -am"comments"*

Hacer un stash pero sin guardar los cambios

1. ***HACEMOS CAMBIOS EN RAMA1***
2. *$git stash*
3. *$git stash drop* = Se borran los cambios hechos.

$git stash es un comando típico cuando queremos experimentar con nuestro proyecto, pero esa experimentación no merece una rama o un *$git rebase*

Git Clean: Limpiar tu proyecto de archivos no deseados

Algunas veces se crean archivos que no son parte de nuestro directorio de trabajo. Estos pueden ser resultado de una compilación o a generación de algún archivo plano.

Al hacer *$git status* se desplegarán los archivos no trackeados o en staging. Por lo que podemos ejecutar:

$git clean –dry-run = Se muestran los archivos a borrar (pero no se borrar aun)

$*git clean -f* = Se borran los archivos

Nota: $*git clean* no borra carpetas, solo archivos (las carpetas y todo lo que este en .gitignore se tiene que borrar manualmente)

Git cherry-pick: Traer commits viejos al head

$git cherry-pick nos permite integrar en una rama un cambio/commit antiguo que esta en otra rama

1. Estando en la rama1 ***MODIFICAMOS EL ARCHIVO**
2. *$git stash* = Guardamos momentáneamente los cambios
3. *$git stash branch rama2* = Movemos los archivos a una rama
4. (Automáticamente estamos en rama2) *$git commit -am"commit 1"*
5. ***SEGUIMOS MODIFICANDO EL ARCHIVO***
6. *$git commit -am"commit 2"*

Supongamos que queremos pasar a rama1 **únicamente** los cambios de commit 1

7. *$git checkout rama1*
8. *$git cherry-pick* codigoDelPrimerCommit

$git cherry-pick no es una buena practica porque no deja rastro en la rama que recibió el cambio.

Comandos de emergencia

Git Reset – Git Reflog

Supongamos que todo el repositorio se arruina de alguna manera, con *$git reflog* podemos visualizar absolutamente toda la historia de nuestros commits (incluyendo ramas que ya no existen). Proceso de recuperación

1. *$git reflog*
2. Copiamos el ultimo commitCode en donde todo estaba en orden
3. *$git reset*

   3.1 *$git reset --SOFT* = Se guarda lo que está en staging

   3.2 *$git reset --HARD* = Se resetea todo, sin convervar los cambios de staging
4. Todo el repositorio regresa al último estado elegido

$git reset –HARD debe ser usado con sabiduría como último recurso cuando todo se rompa.

Reconstruir commits

Reconstruimos un commit cuando cometidos un error ar hacerlo (modificamos mal el archivo o lo documentamos incorrectamente). Proceso:

1. ***MODIFICAR EL ARCHIVO***
2. *$git add file.dom*
3. *$git commit --amend* = --amend pega los cambios al commit anterior

Buscar archivos y commits de Git

$git grep palabraBuscar = Nos muestra los archivos y un extracto de donde aparece la palabra

$git grep -n palabraBuscar = Nos muestra en que línea del archivo aparece la palabra

$git grep -c palabraBuscar = Nos muestra la incidencia de la palabra y en que archivos

$git log -S palabraBuscar = Nos muestra en donde aparece la palabra en los commits