INFORME INVESTIGATIVO SOBRE GITHUB

KAREN DANIELA POVEDA

BRYAN MUÑOZ

BRAYAN RACHEN

SEBASTIAN MOLINA

24/02/2025

**HISTORIA**

GitHub es una interfaz basada en la web que utiliza Git, el software de control de versiones de código abierto que permite que varias personas realicen cambios separados en las páginas web al mismo tiempo. Como señala Carpenter, debido a que permite la colaboración en tiempo real, GitHub alienta a los equipos a trabajar juntos para crear y editar el contenido de su sitio.

La idea de GitHub surgió en 2005 con el ingeniero de software finlandés-estadounidense Linus Torvalds, quien también había inventado el sistema operativo del núcleo Linux. Torvalds buscaba un nuevo sistema para mantener Linux y creó Git (en poco más de una semana) como solución

Github fue desarrollado por Chris Wanstrath, P. J. Hyett, Tom Preston-Werner y Scott Chacon usando el framework Ruby on Rails, y empezó en 2008 pese a que la compañía Github, Inc fue registrada en 2007.

El 25 de julio de 2010, GitHub anunció que había llegado al millón de repositorios. Llegando a los 2 millones el 20 de abril de 2012. El 16 de enero de 2013, sobrepasaron los 3 millones de usuarios registrados y alojaban ya más de 5 millones de repositorios, llegando a los 10 millones el 23 de diciembre de 2013. En junio de 2015, GitHub abrió su primera oficina fuera de Estados Unidos, en Japón. En junio de 2015, GitHub abrió su primera oficina fuera de Estados Unidos, en Japón.

**COMANDOS PARA GITHUB**

Configuración básica

* Configurar Nombre que salen en los commits

git config --global user.name "dasdo"

* Configurar Email

git config --global user.email [dasdo1@gmail.com](mailto:dasdo1@gmail.com)

* Marco de colores para los comando

git config --global color.ui true

iniciando repositorio

* Iniciamos GIT en la carpeta donde esta el proyecto

git init

* Clonamos el repositorio de github o bitbucket

git clone <url>

* Añadimos todos los archivos para el commit

git add.

* Hacemos el primer commit

git commit -m "Texto que identifique por que se hizo el commit"

* subimos al repositorio

git push origin master

Git clone

* Iniciamos GIT en la carpeta donde está el proyecto

git init

* Clonamos el repositorio de github o bitbucket

git clone <url>

* Añadimos todos los archivos para el commit

git add.

* Hacemos el primer commit

git commit -m "Texto que identifique por que se hizo el commit"

* subimos al repositorio

git push origin master

Git add

* Añadimos todos los archivos para el commit

git add.

* Añadimos el archivo para el commit

git add <archivo>

* Añadimos todos los archivos para el commit omitiendo los nuevos

git add --all

* Añadimos todos los archivos con la extensión especificada

git add \*.txt

* Añadimos todos los archivos dentro de un directorio y de una extensión especifica

git add docs/\*.txt

* Añadimos todos los archivos dentro de un directorio

git add docs/

Git commit

* Cargar en el HEAD los cambios realizados

git commit -m "Texto que identifique por que se hizo el commit"

* Agregar y Cargar en el HEAD los cambios realizados

git commit -a -m "Texto que identifique por que se hizo el commit"

* De haber conflictos los muestra

git commit -a

* Agregar al ultimo commit, este no se muestra como un nuevo commit en los logs. Se puede especificar un nuevo mensaje

git commit --amend -m "Texto que identifique por que se hizo el commit"

GIT PUSH

* Subimos al repositorio

git push <origien> <branch>

* Subimos un tag

git push –tags

GIT LOG

* Muestra los logs de los commits

Git log

* Muestras los cambios en los commits

git log --oneline --stat

* Muestra graficos de los commits

git log --oneline –graph

GIT DIFF

* Muestra los cambios realizados a un archivo

git diff

git diff –staged

GIT HEAD

* Saca un archivo del commit

git reset HEAD <archivo>

* Devuelve el ultimo commit que se hizo y pone los cambios en staging

git reset --soft HEAD^

* Devuelve el ultimo commit y todos los cambios

git reset --hard HEAD^

* Devuelve los 2 ultimo commit y todos los cambios

git reset --hard HEAD^^

* Rollback merge/commit

git log

git reset --hard <commit\_sha>

GIT REMOTE

* Agregar repositorio remoto

git remote add origin <url>

* Cambiar de remote

git remote set-url origin <url>

* Remover repositorio

git remote rm <name/origin>

* Muestra lista repositorios

git remote -v

* Muestra los branches remotos

git remote show origin

* Limpiar todos los branches eliminados

git remote prune origin

GIT BRANCH

* Crea un branch

git branch <nameBranch>

* Lista los branches

git branch

* Comando -d elimina el branch y lo une al master

git branch -d <nameBranch>

* Elimina sin preguntar

git branch -D <nameBranch>

GIT TAG

* Muestra una lista de todos los tags

git tag

* Crea un nuevo tag

git tag -a <verison> - m "esta es la versión x"

GIT REBASE

* Los rebase se usan cuando trabajamos con branches esto hace que los branches se pongan al día con el master sin afectar al mismo

Une el branch actual con el mastar, esto no se puede ver como un merge

git rebase

* Cuando se produce un conflicto no das las siguientes opciones:

cuando resolvemos los conflictos --continue continua la secuencia del rebase donde se pauso

git rebase --continue