AURA ANDREA MOREJON SILIEZAR

FUNDAMENTOS DE PROGRAMACION Y SCRIPTING

21 DE FEBRERO DEL 2023

GIT

GIT es un sistema de control de versiones, usado en su mayoría por desarrolladores profesionales, facilitando el trabajo en equipo, revisiones del código según su calidad y pasar pruebas, se pueden hacer despliegue de aplicaciones web con integración continua y despliegue continuo.

La instalación de GIT se puede realizar en Linux, Windows y Mac; se pueden crear carpetas en el sistema de archivos con código fuente llamados **repositorios**. Git tiene la característica de ser un sistema de control de versiones con repositorios distribuidos, es decir, que cada uno de los integrantes de un proyecto tiene una copia exacta (un clon) del repositorio y no solamente una copia de los archivos con los que haya trabajado. Los repositorios habitualmente se encuentran en repositorios remotos, en un servidor de Internet, donde los desarrolladores pueden enviar cambios y sincronizar su repositorio local con el remoto. Git también tiene otros servicios de hosting de repositorios Git como Gitlab o Bitbucket.

Borrar una carpeta .git, significa perder toda la gestión del repositorio local y el control de lo realizado, incluso eliminar el proyecto por completo no seria tanto problema si este repositorio esta subido a GitHub. El hosting de repositorios remotos nos asegura que los cambios se mantendrían salvaguardados en cualquier momento podríamos volver a descargar el repositorio, mediante la operación de clonado.

CONTROL DE VERSIONES

Un control de versiones es un sistema que almacena el registro de los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, teniendo este registro podemos recuperar versiones especificas cuando lo requiera. Podemos encontrar diferentes sistemas para control de versiones, por ejemplo:

- Sistema de Control de Versiones Locales (VCS): Esta herramienta funciona guardando conjunto de parches (las diferencias entre archivos) en un formato especial en disco, y es capaz de recrear como era un archivo en cualquier momento a partir de dichos parches.
- Sistema de Control de Versiones Centralizado (CVCS): tienen un único servidor donde se encuentran todas las versiones de archivos y se pueden descargar desde ese lugar centralizado, esta estructura ha sido estándar para el control de versiones por muchos años.
- Sistema de Control de Versiones Distribuidos (DVCS): los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas colaboran a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo.

ESTADOS DE UN ARCHIVO

Comprende en un conjunto de comando donde nos muestra el estado del directorio de trabajo y del área del entorno de ensayo, estos comandos no muestran información relativa al historial del proyecto, lo que ha sucedido con los comandos ***git add*** y ***git commit***.

- ***git status*** Enumera los archivos que se **han preparado**, los que están **sin preparar** y los archivos **sin seguimiento.**

COMO SE CONFIGURA UN REPOSITORIO

inicio

- ***git init*** es el primer comando usado para la configuración inicial del repositorio nuevo, primero se establece el directorio de la carpeta raíz del proyecto, al ejecutar este comando, se creara un nuevo subdirectorio .git en tu directorio de trabajo actual.

  cd /path/to/your/existing/code
  git init

  - Archivo /etc/gitconfig: Contiene valores para todos los usuarios del sistema y todos sus repositorios. Si pasas la opción --system a git config, lee y escribe específicamente en este archivo.
  - Archivo ~/.gitconfig o ~/.config/git/config: Este archivo es específico de tu usuario. Puedes hacer que Git lea y escriba específicamente en este archivo pasando la opción --global.
  - Archivo config en el directorio de Git (es decir, .git/config) del repositorio que estés utilizando actualmente: Este archivo es específico del repositorio actual.

Identidad

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los "commits" de Git usan esta información, y es introducida de manera inmutable en los commits que envías:

- $ git config --global user.name "John Doe"
- $ git config --global user.email johndoe@example.com

Tu Editor

Ahora que tu identidad está configurada, puedes elegir el editor de texto por defecto que se utilizará cuando Git necesite que introduzcas un mensaje. Si no indicas nada, Git usará el editor por defecto de tu sistema, que generalmente es Vim. Si quieres usar otro editor de texto como Emacs, puedes hacer lo siguiente:

- $ git config --global core.editor Emacs

COMANDOS GIT

Configurar Email

```
git config --global user.email dasdo1@gmail.com
```

Marco de colores para los comando

```
git config --global color.ui true
```

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
```

Clonamos el repositorio de github o bitbucket

```
git clone <url> git-demo
```

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 directorios

```
git add docs/
```

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

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

Subimos al repositorio

```
git push <origien> <branch>
```

Subimos un tag

```
git push --tags
```

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
```

Muestra los cambios realizados a un archivo

```
git diff

git diff --staged
```

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^
```

Rollback merge/commit

```
git log

git reset --hard <commit_sha>
```

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
```

Crea un branch

```
git branch <nameBranch>
```

Lista los branches

```
git branch
```

Elimina sin preguntar

```
git branch -D <nameBranch>
```

Muestra una lista de todos los tags

```
git tag
```

Lista un estado actual del repositorio con lista de archivos modificados o agregados

```
git status
```

Quita del HEAD un archivo y le pone el estado de no trabajado

```
git checkout -- <file>
```

Crea un branch en base a uno online

```
git checkout -b newlocalbranchname origin/branch-name
```

Cambiar de branch

```
git checkout <nameBranch/tagname>
```

Une el branch actual con el especificado

```
git merge <nameBranch>
```

Verifica cambios en el repositorio online con el local

```
git fetch
```

Borrar un archivo del repositorio

```
git rm <archivo>
```