GIT: REPOSITORIO LOCAL

GIT: Es un software de control de versiones, pensado para la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran numero de archivo de código fuente. Básicamente, permite estandarizar la información y que todos estén hablando en el mismo tono. Por ej, si uno realiza un cambio, este programa se asegurará de que los cambios queden implementados para todos los que estén trabajando en lo mismo.

**LA CARPETA. GIT SIRVE PARA RESGUARDAR DETERMINADA INFORMACIÓN/CÓDIGO Y DE ESA MENERA, PODEMOS VOLVER ATRÁS SI LOS CAMBIOS QUE APLICAMOS NO SON LOS MÁS ADECUADOS. GUARDA LA VERSIÓN ANTERIOR**

Comandos más utilizados:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

ORDEN DE LOS COMANDOS:

Creamos el directorio

1. 

Agregamos el archivo. Añade un cambio del directorio de ensayo

1. 
2. Verificamos el status

Texto

Descripción generada automáticamente

Ponemos un mensaje identificatorio de lo que estamos haciendo. Este comando captura una instantánea de los cambios preparados en ese momento del proyecto, las mismas pueden considerarse una versión segura del proyecto

1. Interfaz de usuario gráfica, Texto, Sitio web

   Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Para configurar el git, ponemos primeramente:



Esa es la estructura general, sino podemos escribir: git –help y nos tira la siguiente ayuda según el comando:

**En la ayuda de git tenemos:**

$ git --help

usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

[--super-prefix=<path>] [--config-env=<name>=<envvar>]

<command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.

GIT PUSH

Manda todo lo que tenemos en nuestro repositorio local, a nuestro repositorio remoto

GIT PULL

Con esto lo que hacemos, es descargar contenido de un repositorio remoto a nuestro directorio local y mantener en la misma sintonía el remito y el local

GIT STATE

Muestra el estado actual de todo el árbol de trabajo.

ESQUEMA DE TRABAJO

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Acá vemos como quedaría un trabajo que se va modificando y guardando en git. Las líneas son ramas del árbol, y los desvíos son las diferentes actualizaciones que fue sufriendo o ampliándose

Como definimos entonces un proyecto de ciencia de datos:

Estrategia de branching: creación y desarrollo de ramas para trabajar en equipo durante un proyecto. Para evitar que se vayan pisando en el trabajo.

Gráfico

Descripción generada automáticamente

Master: sería nuestro tronco del árbol en nuestro proyecto

Hotfix: es un arreglo que se tuvo que hacer en forma rápida

Develop: desarrollo, donde se trabaja siempre

Release: versión beta del desarrollo original. Sale a producción, va a usar la persona que pidió ese desarrollo.

Feature: se agregan cosas al desarrollo

Rama: /Branch

La creación de ramas es una función disponible en la mayoría de los sistemas de control de versiones modernos. Cuando quieres añadir una nueva función o solucionar un error, ind, de su tamaño, generas una nueva rama.

Merge:

Es un comando que permite retroceder a determinada etapa sino habíamos guardado los cambios correctamente.

GIT LOG: Muestra todas las coomits en el historial del repositorio

GIT CHECKOUT: Cambia entre ramas o restaura los archivos del árbol de trabajo.

**USO DE GITHUB**

**Es un sistema web que utiliza GIT para ofrecer a la comunidad de desarrolladores repositorios de software, es decir, es un sitio web pensado para hacer posible el compartir el código de una manera mas fácil y al mismo tiempo darl popularidad a la herramienta de control de versiones entre si**

**EN GITHUB PODEMOS FORKEAR PROYECTOS, OSEA, LLEVARME PROYECTOS DE OTROS A MI REPOSITORIO**

Texto

Descripción generada automáticamente

Luego de realizar el fork, lo tenemos que clonar en nuestro directorio local. La sentencia es la siguiente:



1.32.47

AHORA VAMOS A CONFIGURAR UNA COPIA DESDE EL TRAER UNA NUEVA RAMA QUE VA A SURGIR DEL TRONCO PRINCIPAL CON EL COMANDO “cd”

