

Tutorial de GIT

Introducción

Git es la herramienta que nos permite tener un historial completo de todo el código que hemos desarrollado en nuestra aplicación.

Supongamos que nuestro código es la siguiente aplicación en python:

```
edad = int(input("ingresa tu edad: "))          Versión 1.0


if edad >=18:
    print("Sos mayor de edad")
else:
    print("Sos menor de edad")
```

Esta podría ser la versión 1.0 de nuestro programa. Luego necesitamos agregarle una nueva funcionalidad, le agregamos más código y ahora ya tenemos la versión 1.1.

```
edad = int(input("ingresa tu edad: "))          Versión 1.1

if edad >=18:
    print(f"{nombre}, tenes {edad} años y sos  
mayor de edad")
else:
    print(f"{nombre}, tenes {edad} años y sos  
menor de edad")
```



Pero al agregarle nuevo código puede sucedernos que introducimos involuntariamente errores dentro de nuestra aplicación. Ahora nuestra aplicación tiene un bug 

Ahora lo que debemos hacer es un rollback, es decir volver de la versión 1.1 la cual contiene el error a la 1.0 la cual sabemos que no tenía errores.

Pero sino utilizamos un sistema de control de versiones como Git, la versión 1.0 ya no existe, excepto que hayas guardado una copia por ahí. En este caso tenemos dos opciones:

La primera es solucionar el error y seguir agregando código a nuestra versión 1.1.

La segunda opción es tratar de acordarnos como era la versión 1.0 y llevar el código a esta versión.

Estas dos posibles soluciones podrían introducir nuevos errores en nuestro código. Ahora imagina este mismo problema pero trabajando en grupo con una multitud de programadores, donde todos se encuentran realizando cambios sobre el código. Sería muy caótico y problemático, pero para ello existe Git.

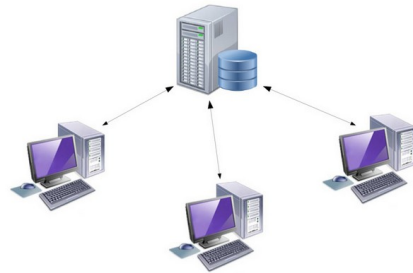
Tipos de sistemas de control de versiones

Centralizados

Los programadores deben actualizar los cambios en un servidor que se encuentra en Internet.

Si el servidor deja de funcionar, los programadores no van a poder obtener los cambios que realizaron otros programadores y tampoco van a poder subir sus cambios.

- Existe un servidor centralizado que almacena el repositorio completo
- La comunicación/colaboración entre desarrolladores se lleva a cabo (forzosamente) utilizando el repositorio centralizado
- Son más simples de usar
- Los modelos de trabajo son más restringidos

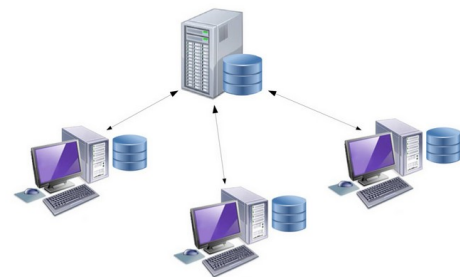


Distribuidos

Git trabaja de manera distribuida, significa que cada uno de los programadores tiene una copia completa del código que se está trabajando. El programador tiene la opción de subir los cambios realizados localmente a un servidor central, de esta manera si sincronizan los cambios entre todos los programadores.

Si por alguna razón el servidor remoto dejara de funcionar, no afectaría tanto el trabajo en grupo, ya que cada uno de los miembros ya contaría con una copia del código dentro de su computadora, por lo tanto van a poder seguir agregando código.

- Cada desarrollador contiene una copia completa de todo el repositorio
- Los mecanismos de comunicación/colaboración entre desarrolladores son más abiertos
- Son (un poco) más difíciles de utilizar que los sistemas centralizados
- Los modelos de trabajo son más flexibles
- “Los branches/merges son más simples”



Usos de Git

- Historial de todo el trabajo que hemos realizado sobre nuestro código y el historial de trabajo que otros programadores han realizado sobre el mismo código.
- Almacenar código.
- Trabajar en equipo de una manera fácil y simple. Git se va a encargar de transmitir los cambios que los programadores les realizan a los distintos archivos.
- Saber cuando se introdujo un error en una aplicación. Cuando encontramos un error podemos volver atrás y ver dentro de nuestro historial y encontrar que commit fue el que introdujo cierto error y de esta manera reducir la cantidad de código que debemos revisar para poder encontrar dicho error.

Cliente de Git

Existen herramientas gráficas para usar Git, pero usaremos la terminal porque es la forma más habitual y más poderosa de utilizar Git. Además la mayoría de los programadores con los que trabajarás utilizarán la terminal.

Instalación de Git. Lo debes descargar desde <http://git-scm.com/>
Es muy fácil de instalar.

La versión windows además trae la aplicación Git BASH. Git bash nos permitirá ejecutar los comandos de Git y además emulará los comandos de Linux.

No utilices la terminal de Windows, utiliza Git-Bash. Respeta esto y te ahorrarás dolores de cabeza con Git.

Si instalamos Git en Linux o Mac, veremos que Git no se muestra al igual que en Windows, para que se muestre similar a Windows tenemos varias maneras, te recomiendo una forma sencilla de utilizar Git similar a windows. Para Linux o MacOS instala Oh-my-zsh.

Configuración de Git (git config)

git config	--global	Archivo de configuración para todos los repositorios del usuario
	--local	Archivo de configuración para un repositorio
	--system	Archivo de configuración para todo el sistema

git config --global	user.name	"Juan Perez"	Nombre y apellido del usuario
	user.email	juanperez@gmail.com	E-mail del usuario
	core.editor	"code --wait"	Visual Studio Code (debemos instalarlo previamente, sino utilizará el editor por omision)
	core.autocrlf	true	Para usuarios de windows debe ser true. Para Linux y Mac input.
	-e	Editar el archivo de configuración global	
git config -h	Ayuda de la configuración		

Editar el archivo de configuración global

git config --global -e

```
[user]
  email = arielmpereira@gmail.com
  name = Ariel Pereira
[core]
  editor = vim
  autocrlf = input

~/ .gitconfig 6L, 101C 6, 17-24 Todo
```

Resumen de comandos de la consola git-bash

Estos son comandos de bash, no realizan ninguna accion en git, pero nos serán util para movernos con la consola por el sistema de archivos.

ls	Listado de archivos y directorios dentro del directorio actual
pwd	“Print working directory” muestra el directorio actual, donde nos encontramos
cd Documentos	Cambiamos el directorio de trabajo a Documentos
cd ..	Subimos un nivel en el arbol de directorios
mkdir miweb	Crea el directorio “miweb” dentro del directorio actual

Inicializar un repositorio Git

Vamos a crear un directorio (miweb) dentro del directorio actual e inicializar un repositorio Git.

mkdir miweb

Con este comando creamos el directorio miweb.

cd miweb

Con este comando cambiamos el directorio actual por el directorio miweb.

```
→ ~ mkdir miweb
→ ~ cd miweb
→ miweb git init
Iniciado repositorio Git vacío en /home/ariel/miweb/.git/
→ miweb git:(master) ls
→ miweb git:(master) □
```

Ahora aparece:

→ miweb

Nos indica que estamos dentro del directorio miweb

Inicializamos el repositorio git:

git init

Ahora ya tenemos el repositorio inicializado y vemos que aparece:

→ miweb git:(master)

Como en ese directorio existe un repositorio de Git, van a aparecer los mensajes de Git. Si salimos del directorio, ya no aparecerán más los mensajes de Git.

Git cuando inicializa un repositorio, crea un directorio `.git` que contiene el repositorio y la configuración. Ese directorio comienza con un punto indicando que es un directorio oculto, por eso cuando ejecutamos el comando `ls`, no aparece nada, como si estuviese vacío, pero en realidad hay un directorio oculto (`.git`).