Explicación práctica de GIT

Explicación Práctica - Seminario de Lenguajes Opción Python

Equipo de desarrollo de software

- ¿Cuántos desarrolladores contribuyen a un **mismo proyecto**?
- ¿Cuántos trabajarán en el **mismo horario**?
- ¿Cuántos trabajarán en el **mismo lugar**?
- ¿Cuántos trabajarán en un mismo momento realizando modificaciones en un **mismo archivo**?

Desafíos de trabajo colaborativo

- Dependiendo de la magnitud del proyecto podemos encontrarnos de unos pocos pares de desarrolladores hasta cientos o miles realizando contribuciones.

- **Desafíos en compartición:** qué método utilizaría para que todos tengan acceso al mismo código. ¿Carpeta compartida? ¿FTP? ¿Enviar las modificaciones por mail?

Desafíos de trabajo colaborativo

- **Desafíos en la comunicación:** no todos pueden comunicarse entre sí para coordinar una modificación en el código del proyecto.. Además si el equipo es grande sería inmanejable.

- **Desafíos en el control de calidad:** cómo hacer para que sólo las modificaciones que el responsable del desarrollo considera viables sean las que se vean reflejadas en el proyecto.

Desafíos de trabajo colaborativo

- **Desafíos en el desarrollo:** ¿qué haríamos si necesitamos llevar un desarrollo paralelo de una nueva funcionalidad, que necesita estar actualizado con los nuevos cambios que se realicen en el proyecto principal?

Git como solución a estos problemas

GIT es un sistema de versionado de código que permite a un equipo de desarrollo trabajar de forma **sincronizada** dando las herramientas para **evitar la pérdida o sobreescritura de código**. Además mantiene un **historial de todos los cambios** que se realizaron en el repositorio y nos posibilita deshacer cualquier cambio realizado en el mismo.

Git como solución a estos problemas

- **Cantidad de desarrolladores:** Ilimitada. Complejidad manejable con buenas prácticas.
- **Desafíos de compartición:** Todos tienen acceso al mismo código y a sus diferentes ramas de desarrollo.
- **Desafíos en la comunicación:** Los desarrolladores pueden saber quién, cuándo y por qué hizo determinado cambio.

Git como solución a estos problemas

- **Desafíos en el control de calidad:** herramientas como GitLab y GitHub, por ejemplo, nos proveen los **Pull Requests**, con lo que las modificaciones sólo serán agregadas al código principal sólo si cumplen con los requisitos que fije quien deba aprobarlo.

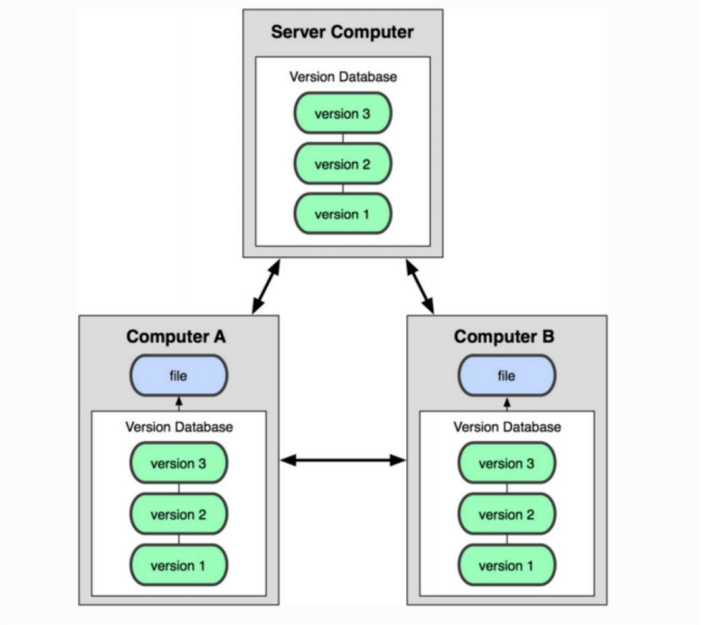
- **Desafíos en el desarrollo:** gracias a las Ramas o Branches, Git nos permite llevar desarrollos en paralelo sin dejar de tener el código sincronizado con el proyecto principal.

Sistema de control de versiones distribuido

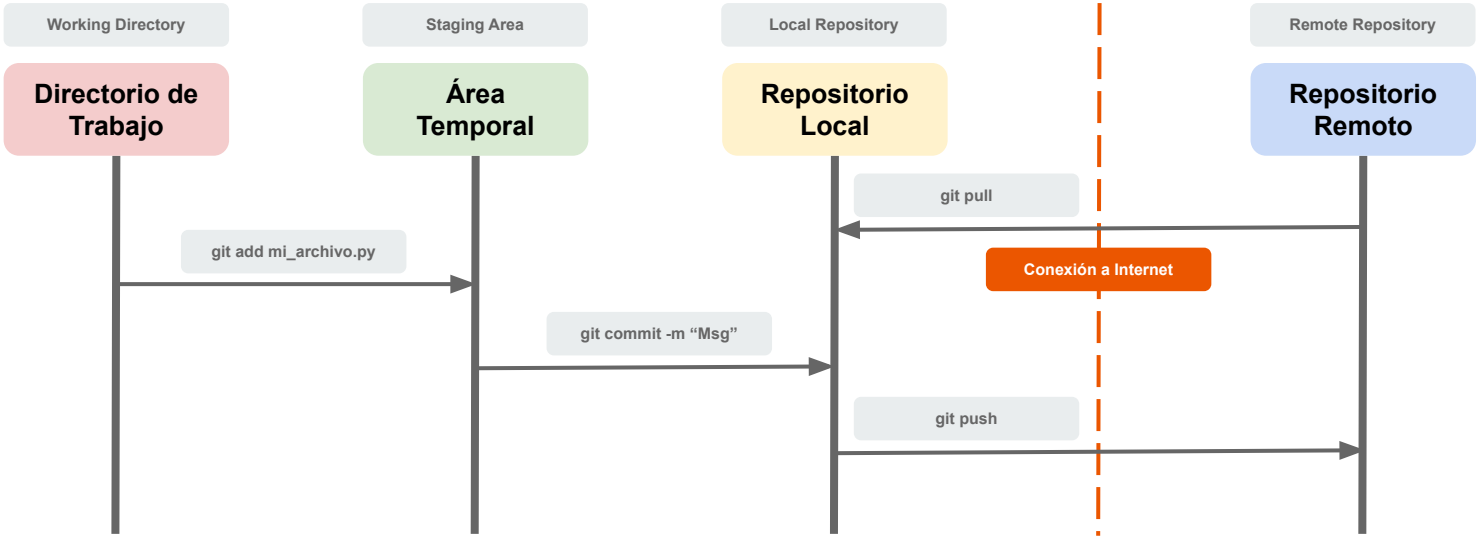
- **Historial de cambios** de uno o varios archivos
  - Permite **recuperación** de versiones anteriores
- Ante **modificaciones simultáneas** provee herramientas de **resolución de conflictos**
- **Copias locales** del repositorio remoto
  - Permite la posibilidad de **trabajar sin conexión**

Sistema de control de versiones distribuido

- **Arquitectura Cliente-Servidor**
  - 1+ clientes
  - 1+ servidores o **remotes (usualmente solo 1)**
- El servidor es quien posee la última versión del código centralizado.
- Ciertos clientes pueden estar:
  - **Detrás:** si no sincronizan con el servidor y otro cliente envió cambios.
  - **Delante:** si el cliente realizó cambios en su repositorio local pero aún no los envió al servidor.



Estadíos de Git



Requerimientos para trabajar con GIT

- Tener instalado el cliente de GIT en el SO.
  - Cómo saber si está instalado: tipear en la terminal `git -v`
  - Instalación en windows: https://git-scm.com/download/win
- Tener acceso a algún servidor de GIT y crear el repositorio remoto allí:
  - Github: https://github.com
  - Bitbucket: https://bitbucket.org

Ejercicio práctico en repositorio local PC 1

- Abrimos la terminal
- Nos dirigimos a la carpeta home de nuestro usuario: `cd`
- Creamos una carpeta para nuestro proyecto: `mkdir ejemplo_git`
- Entramos a la carpeta: `cd ejemplo_git`
- Inicializamos el repositorio: `git init`
- Configuramos nuestra identidad:
    - `git config user.name usuario1`
    - `git config user.email usuario1@mailinator.com`

Ejercicio práctico en repositorio local PC 1

- Creamos un archivo: nano main.py
    - print(“¡Hola! estoy usando GIT.”)
- Agregamos los cambios al área de staging: git add main.py o git add --all
- Verificamos los cambios hechos: git status
- Hacemos commit de los cambios: git commit -m “Mi primer archivo”
- Verificamos el historial: git log o git log --oneline

Ejercicio práctico en repositorio remoto PC 1

- Creamos una cuenta en Github con el mismo email que configuramos nuestro repositorio local.
- Creamos un nuevo repositorio en Github.
- Copiamos la dirección de nuestro repositorio remoto.
- Lo agregamos como **origin**: `git remote add origin [user@github.com:repo-name]`

Ejercicio práctico en repositorio remoto PC 1

- Si no tenemos una clave ssh la creamos: `ssh-keygen` (seguimos los pasos del wizard hasta generarla)
- Agregamos la clave ssh a nuestro cliente: `ssh-add ~/.ssh/id_rsa`
- Copiamos el contenido de la clave pública: `cat ~/.ssh/id_rsa.pub`
- Agregamos la clave en nuestra cuenta de Github (profile avatar > settings > SSH and GPG keys > New ssh key) **GitLab** funciona de la misma forma
- Enviamos los cambios de nuestro repositorio local: `git push origin master`
- Agregar usuario nuevo al repositorio, pueden invitarlo vía github y le enviará un email con el link para ingresar al proyecto.

Ejercicio práctico en repositorio remoto PC 2

- Ingresamos a nuestra casilla de correo y aceptamos la invitación.
- Creamos la clave ssh y la agregamos a nuestro perfil de Github.
- Ingresamos a la terminal y vamos a la carpeta donde queremos alojar nuestro proyecto: `cd ~/proyectos/python`
- Comprobar si tenemos git instalado, si no es así, instalarlo.
- Clonamos el repositorio: `git clone git@github.com:[repository_path]`

Ejercicio práctico en repositorio remoto PC 2

- Agregamos una nueva línea de código en el archivo main.py

  print("¡Hola! estoy usando GIT.")
  print("¡Yo también!")
- Verificamos los cambios: git status
- Agregamos los cambios: git add --all
- Enviamos cambios al repositorio local: git commit -m "Agregada nueva línea"
- Enviamos cambios al repositorio remoto: git push origin master

Ejercicio práctico en repositorio remoto PC 1

- Agregamos una nueva línea de código en el archivo main.py

  print("¡Hola! estoy usando GIT.")
  print("Ahora no sé qué más agregar...")
- Verificamos los cambios: git status
- Agregamos los cambios: git add --all
- Enviamos cambios al repositorio local: git commit -m "Agregada nueva línea"
- Git nos dice que no podemos enviar los cambios porque debemos sincronizar con el remoto.
- Sincronizamos: git pull origin master

Ejercicio práctico en repositorio remoto PC 1

- Conflicto en el archivo main.py
- Abrimos el archivo: nano main.py
- Dentro del archivo verán algo así:

```
print("¡Hola! estoy usando GIT.")
<<<<<<< HEAD
print("Ahora no sé qué más agregar...")
=======
print("¡Yo también!")
>>>>>>> hash del commit que realizó el usuario de la PC 2
```

Ejercicio práctico en repositorio remoto PC 1

- Resolvemos el conflicto a mano
  - Mantenemos nuestra modificación y descartamos la conflictiva 💩
  - Mantenemos la modificación conflictiva y descartamos la nuestra😶
  - Mantenemos ambas modificaciones 🙌

💬 Esto requiere **comunicación con el equipo de desarrollo**. Podemos, a través del hash del commit conflictivo, averiguar quién es el usuario que lo envió. Hablemos con él antes de tomar una decisión y coordinemos el merge.

Uso avanzado de GIT e introducción a GitLab

Guía de GIT y GitLab:

https://bit.ly/2R0WyVX