**Git – GitLab**

* Sistema de control de versiones:
  + Es un sistema que registra los cambios realizados a nuestros archivos en el tiempo, de modo de poder volver a una versión anterior en cualquier momento.
  + Sistema de control de versiones local:
    - Se comenzaron a desarrollar utilidades para manejar las revisiones localmente en una BD.
    - Todas estas operaciones son locales, compartir el trabajo con otro desarrollador era difícil.
  + Sistema de control de versiones centralizado:
    - Las revisiones se tienen en un servidor centralizado.
    - Los desarrolladores deben descargar y subir las nuevas versiones para compartirlas.
  + Sistema de control de versiones distribuido:
    - Cada desarrollador tiene una copia de todo el repositorio, cada uno trabaja a su ritmo.
    - Como la mayoría de las operaciones son ahora locales y no necesitan red, la velocidad de desarrollo se incrementó.
* Git:
  + Es un sistema de control de versiones distribuido libre diseñado para manejar proyectos con velocidad y eficiencia.
  + Características:
    - Snapshots, no diferencias:
      * Modela sus datos más como un conjunto de instantáneas (snapshots) de un mini sistema de archivos.
      * Por eficiencia, si un archivo no cambio, no vuelve a guardarlo, solo referencia al archivo ya almacenado.
    - Casi todas las operaciones son locales.
      * La mayoría de las operaciones en Git solo necesitan archivos y recursos locales para operar.
      * Para navegar por la historia del proyecto, Git no necesita buscarla en el servidor.
    - Tiene integridad.
      * Todo en Git es verificado mediante una suma de comprobación antes de ser almacenado, y es identificado a partir de ese momento mediando dicho checksum.
      * Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa.
  + *git init*: para inicializar un repositorio en un directorio existente.
  + Directorio .git/:
    - Cada repositorio Git es almacenado en la carpeta .git del directorio en el cual el repositorio ha sido creado.
    - Este directorio contiene la historia completa del repositorio. El archivo .git/config contiene la configuración local del repositorio.
  + Operaciones locales:
    - Git tiene tres estados principales en los que se pueden encontrar los archivos:
      * Committed (Confirmado).
      * Modified (Modificado).
      * Staged (Preparado).
    - *git status*: para saber el estado del directorio.
    - *git info:* para ver toda la información del repositorio.
    - *git add* . : para preparar archivo para el repositorio: (con el punto estamos diciendo que agregue TODO).
    - *git commit –m “nombre de referencia”: a*gregar y/o confirmar archivo en el repositorio.
    - *git log*: mostrar un historial del repositorio:.
    - Ciclo de vida de commits:
      * 
    - *git diff* : ver los cambios realizados desde el ultimo commit.
    - *git checkout*:
      * Este comando saca contenido del repositorio y lo pone en el área de trabajo.
      * De esta forma permite descargar los cambios no commiteados (los agregados con git add al área de staging).
      * Además puede utilizarse para “moverse” de rama.
      * Este comando NO altera la historia de commits.
    - *Git revert*:
      * Este comando crea un nuevo commit que deshace los cambios introducidos por un commit previo.
      * Agrega nueva historia al proyecto, NO modifica lo existente.
    - *Git reset:*
      * Dependiendo de cómo es utilizado realiza operaciones muy distintas.
      * Modifica el área de staging.
      * Permite modificar que commit es el último (head) de una rama.
      * Este comando altera la historia del repositorio.
    - Cuanto utilizar cada comando, *checkout, revert o reset:*
      * Si modificamos erróneamente un archivo en nuestra área de trabajo sin haber commiteado los cambios, se debe utilizar *git ckeckout* para traer una copia anterior del archivo desde el repositorio local.
      * Si en la historia del proyecto se produjo un commit que luego se decide que fue erróneo, se debe utilizar *git revert*. Va a deshacer los cambios provocados por el commit erróneo, generando un nuevo commit que registra esta corrección.
      * Si realizamos un commit erróneo, pero no lo hemos compartido con nadie más (todos los commits son locales). Se puede utilizar un *git reset* para reescribir la historia como si nunca se hubiera realizado el commit con errores.
  + Conflictos:
    - Si al actualizar el repositorio local con el remoto (*git fetch + git merge = git pull*) no se puede realizar el merge por estar las mismas líneas modificada se produce un CONFLICTO.
    - Si estos ocurren debemos editar el archivo en conflicto, el cual va a tener marcadas las líneas con problemas y luego volverlo a agregar con git add y git commit.
  + Archivos .gitignore:
    - Sirven para definir, archivos o directorios que serán ignorados por el sistema de control de versiones ya que no todos los archivos dentro de nuestro directorio de trabajo deben ser versionados.
    - Que debemos ignorar:
      * Archivos con configuraciones o información sensible que no debe ser pública.
      * Dependencias de la app.
      * Archivos generados por la app como logs, cache, archivos subidos, binarios, dumps, etc.
  + Ramas (Branches):
    - Es una línea de desarrollo independiente. Uno puede desarrollar una nueva funcionalidad independientemente sin interferir con la línea principal.
    - Por defecto *git init* crea una rama master para trabajar.
    - *Git branch “nombre”* : es para crear una nueva rama.
    - *Git checkout “nombre”* :para cambiar el directorio de trabajo a esa rama.
    - *Git branch* –l: listado de ramas locales.
    - *Git branch* –r: listado de ramas remotas.
    - *Git branch* –a: todas las ramas.
  + Etiquetas (tags):
    - Git nos da la posibilidad de marcar o taggear puntos específicos dentro de la historia de nuestro repositorio. Lo más común es marcar las versiones del desarrollo.
    - *Git tag:* listado de tags.
    - Creando tag liviano:
      * Los tags livianos son como ramas que nunca cambian, punteros a commits.
      * Ejemplo: *Git tag v1.4-lw*
    - Creando tag anotado:
      * Los tasg anotados, en cambio, son guardados como un objeto completo en git. Se le aplica un checksum conteniendo el nombre del autor, su mail y la fecha, tiene un mensaje y pueden ser firmados y verificados con GNU Privacy Guard (GPG).
      * Ejemplo: *git tag –a v1.4 –m “my versión 1.4”*
  + GitFlow:
    - Es un flujo de trabajo basado en ramas (branches) propuesto por Vicent Driessen en 2010.
    - Propone una serie de “reglas” para organizar el trabajo en equipo.
    - Reglas 🡪 Dos ramas principales:
      * Master: cualquier commit que pongamos en esta rama debe estar preparado para subir a producción.
      * Develop: rama en la que está el código que confirmara la siguiente versión planificada del proyecto.
      * Cada vez que se incorpora código a master, tenemos una nueva versión.
    - Ramas Auxiliares:
      * Feature: se originan e incorporan siempre a develop, son las nuevas características de la app.
      * Release: se originan en develop y se incorporan a master y develop. Se utilizan para preparar el siguiente código en producción.
      * Hotfix: se originan en master y se incorporan a master y develop. Se utilizan para corregir errores y bugs en el código en producción.
      * Estas ramas suelen desaparecer una vez incorporadas.
* GitLab:
  + Es una aplicación opensource que nos permite administrar repositorios en git mediante una interfaz web.
  + Clon de GitHub y es una herramienta muy potente para el desarrollo.
  + GitLab utiliza claves SSH para permitir trabajar con los repositorios.
  + Las claves son utilizadas para establecer una conexión segura entre los repositorios y GitLab.
  + Con lo cual lo primero que necesitamos hacer es subir nuestra clave pública al proyecto.