«Talento Tech»

Back-End

Java

Clase 09

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase 9: Introducción a Git y control de versiones

Índice

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Objetivos de la clase

- Comprender el propósito y los beneficios del control de versiones en proyectos de software.
- Instalar y configurar **Git**, la herramienta principal de control de versiones.
- Aprender el flujo de trabajo básico con Git: crear repositorios, añadir cambios al staging (add), registrar cambios con commits y enviar esos commits a un repositorio remoto (push).
- Entender cómo colaborar con otros desarrolladores a través de plataformas como GitHub, utilizando ramas, pull requests y revisiones de código.
- Conocer buenas prácticas de versionado para mantener un historial claro y útil.

Bienvenida a TechLab

Talento Lab

¡Les damos nuevamente la bienvenida a **TechLab**! Hasta ahora, hemos creado un backend sólido con POO, herencia, polimorfismo, manejo de excepciones y organización en paquetes o módulos. Pero a medida que el equipo crece y el código se vuelve más extenso, necesitamos una forma de registrar, compartir y controlar los cambios de manera eficaz, evitando que sobrescribamos el trabajo de los demás y permitiendo volver atrás en el tiempo si algo sale mal.

El **control de versiones** es la respuesta, y **Git** es la herramienta estándar en la industria. Además, plataformas como **GitHub** facilitan la colaboración remota, revisiones de código y la integración continua. Integrar Git al flujo de trabajo de TechLab permitirá un desarrollo más ordenado, transparente y colaborativo.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Conceptos fundamentales de control de versiones

El control de versiones guarda un historial de los cambios realizados en el código a lo largo del tiempo. Esto aporta:

- **Rastreo de cambios:** Saber quién cambió qué y cuándo.
- **Recuperación de versiones anteriores:** Volver a una versión estable si una nueva introdujo errores.
- **Paralelismo:** Permite que varios desarrolladores trabajen en distintas funcionalidades simultáneamente.

Beneficios del control de versiones

- **Seguridad:** Si un archivo se daña, podemos recuperar una versión anterior.
- **Colaboración:** Todas las personas pueden ver qué cambios hizo cada quien y pueden revisar y discutir cada commit.
- **Confianza:** Antes de liberar una versión a la o a el cliente, sabremos exactamente qué commits están incluidos y podremos replicar el estado del proyecto.

Instantáneas del código e historial de cambios

Git no almacena sólo las diferencias, sino instantáneas del proyecto. Cada commit es un estado completo del repositorio en un momento dado. Esto facilita navegar por el historial y comparar versiones.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Instalación y configuración de Git

1. **Instalación:**
   - Visitar https://git-scm.com/downloads y elegir el instalador para tu sistema operativo.
   - Seguir los pasos del instalador con las opciones por defecto, salvo que se requiera algo específico.

2. **Configuración inicial:**

   Configurar el nombre de usuario y email:

   ```
   git config --global user.name "Tu Nombre"
   git config --global user.email "tuemail@example.com"
   ```

3. Esto asocia tus commits con tu identidad.

Comandos básicos iniciales

- `git init`: Crea un repositorio nuevo en una carpeta.
- `git status`: Muestra el estado del repositorio (archivos modificados, etc.).
- `git add`: Mueve archivos modificados al área de preparación (staging).
- `git commit`: Registra cambios en el historial.
- `git log`: Muestra el historial de commits.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Flujo de trabajo básico con Git

El flujo de trabajo típico:

1. **Editás archivos** en tu proyecto.
2. **git add** agrega cambios específicos al área de preparación.
3. **git commit** guarda esos cambios en el historial.
4. **git push** envía tus commits al repositorio remoto (en GitHub u otro servidor).

Creación de repositorios

- `git init` en una carpeta existente la convierte en un repositorio local.
- `git clone URL_DEL_REPOSITORIO` crea una copia local de un repositorio remoto.

Comandos básicos: add, commit, push

```
git add archivo.txt
git commit -m "Descripción breve del cambio"
git push origin main
```

`origin` es el nombre remoto por defecto, `main` es la rama principal (antes era `master`).

Ramas *(branches)* y fusiones *(merges)*

Las ramas permiten trabajar en nuevas funcionalidades sin afectar la estructura (o rama) principal. Por ejemplo, `git branch nueva-funcionalidad` crea una rama. `git checkout nueva-funcionalidad` cambia a ella, y `git merge` integra cambios de una rama en otra.

Colaboración en proyectos mediante GitHub

GitHub (o GitLab, Bitbucket) es un servicio que aloja repositorios remotos. Facilita el trabajo en equipo:

- **Fork:** Crear una copia personal del repositorio.
- **Pull Request:** Proponer cambios al repositorio original.
- **Revisiones de Código:** Comentar y discutir cambios antes de fusionarlos.

Matías y Sabrina podrán trabajar en ramas separadas y, cuando estén listos, crear un Pull Request para que Silvia revise y apruebe los cambios.

Buenas prácticas de versionado

- **Mensajes de commit claros:** Describir brevemente el cambio. Ejemplo: `"Agrega cálculo de descuento para bebidas"`.
- **Commits pequeños y frecuentes:** Facilitan revertir cambios y revisar el historial.
- **Uso de ramas:** Mantener la rama principal estable. Desarrollar nuevas funcionalidades en ramas separadas y fusionarlas cuando estén listas.

Flujo de trabajo ramificado *(branching models)*

Existen modelos como Git Flow, que sugiere ramas dedicadas a desarrollo (`develop`), liberación (`release`), soporte de versiones (`hotfix`), etc. Esto impone orden en el ciclo de vida del software.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Situación inicial en TechLab

Silvia, la Product Owner, nota que Matías y Sabrina empiezan a chocar con un problema: al trabajar en el mismo archivo, uno sobrescribe el cambio de la otra sin quererlo. Además, cuando Silvia pide "¿qué versión del software estaba funcionando bien ayer?" no hay una forma simple de saberlo.

Matías y Sabrina quieren un sistema que les permita:

- Conservar un historial de cada cambio realizado.
- Trabajar en paralelo sin pisarse.
- Unir cambios (merge) cuando estén listos.
- Retroceder a una versión previa si algo va mal.

Esto es exactamente lo que ofrece Git. Con Git y GitHub podrán crear repositorios, hacer commits, ramas, y colaborar de forma ordenada y eficaz.

Para llevar a cabo este desafío debes realizar las siguientes acciones:

Ejercicios Prácticos

Crea un repositorio local

- Creá un repositorio local, registrá cambios con `commit`, trabajá en una nueva rama y fusionala en la rama principal. Finalmente, realizá un `push` al repositorio remoto para compartir los cambios.

Repositorio remoto en Github

- Creá un repositorio remoto vacío en GitHub, e inicializá un proyecto local, enlazalo con dicho repositorio y hacé el primer `push` de la rama principal, verificando que los archivos se reflejen en GitHub.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Materiales y recursos Adicionales

- Documentación Oficial de Git:
- Guía de Git de Atlassian: https://www.atlassian.com/es/git/glossary#commands

Preguntas para Reflexionar

- ¿Cómo mejora la productividad y colaboración en TechLab el uso de Git y GitHub?
- ¿Qué ventajas ofrece mantener un historial detallado de cambios del código?
- ¿Cómo los branching models ayudan a mantener un flujo de trabajo ordenado cuando múltiples desarrolladores colaboran?

Buenos Aires Ciudad